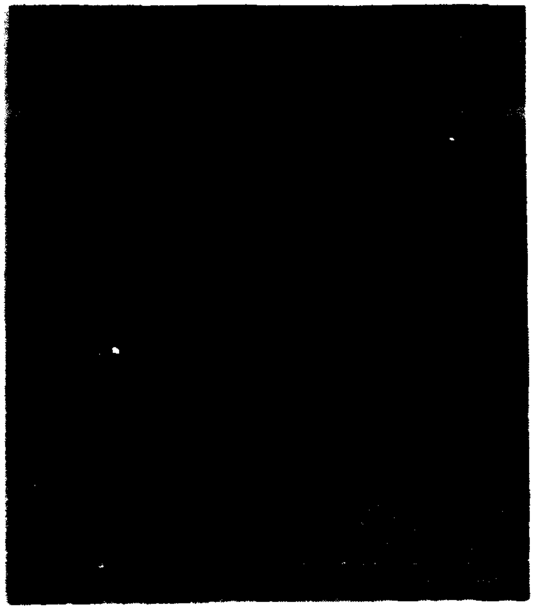
IFIP

INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING

IFIP WG 2.5 Working Conference 6

Programming Environments for High Level Scientific Problem Solving

Karlsruhe
September 23 – 27, 1991



Co-chairs of the Program Committee: Bo Einarsson, Lloyd D. Fosdick
Co-editors: Patrick Gaffney, Elias Houstis
Local Organization: Ulrich Kulisch, Adolf Schreiner

The meeting is co-organized by the Institute for Applied Mathematics of Karlsruhe University.

Thursday, September 26, Afternoon/Evening Session

7 Open Session

(Elias Houstis / Bo Einarsson)

14:00	Johan A van Hulzen	NL	Automated Generation of Optimized Numerical Code for Jacobians and Hessians
14:20	Discussion		
14:30	Niklas Holsti	SF	Transcript Editing, A Simple User Interface Tool
14:50	Discussion		
15:00	Allan Bonadio	USA	Mathematical User Interfaces for Graphical Workstations
15:20	Discussion		
15:30	Break		
16:00	David Gay	USA	Toward an Environment for Mathematical Programming
16:20	Discussion		
16:30	John Reid	UK	The Fortran 90 Standard
16:45	Discussion		
16:50	General Discussion		
17:00	Close		

Demonstrations

19:00	Elias Houstis	USA	Parallel Ellpack (Sun Sparc)
19:20	Siu Shing Tong	USA	Integration of Symbolic and Numerical Methods for Optimizing Complex Engineering Systems (Video tape)
19:30	Dominique Duval	F	Examples of Problem Solving Using Computer Algebra (IBM RT/PC)
19:50	Kevin Broughan	NZ	SENAC: Lisp as a Platform for Constructing a Problem Solving Environment (Sun Sparc)
20:10	Niklas Holsti	SF	Transcript Editing, A Simple User Interface Tool (IBM PC)
20:30	Allan Bonadio	USA	Mathematical User Interface for Graphical Workstations (Macintosh)

The presentation on Friday 09:40 by I A Nicolayev (SU) was replaced by one by J R Rice (US) on research directions of the National Science Foundation (US) with respect to computer science, essentially within the area covered by the conference.

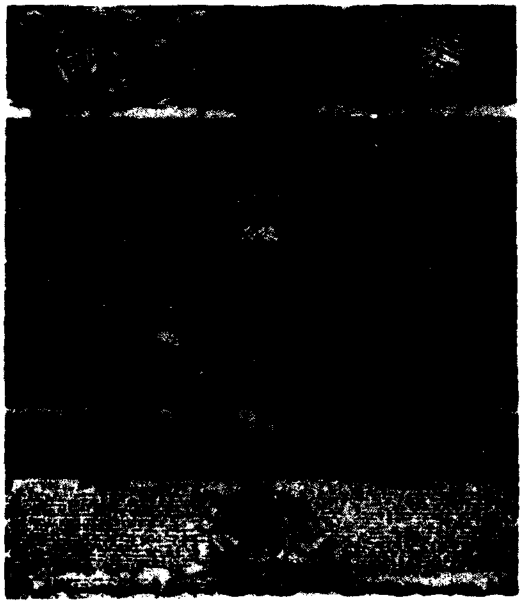
IFIP

INTERNATIONAL FEDERATION FOR INFORMATION PROCESSING

IFIP WG 2.5 Working Conference 6

Programming Environments for High Level Scientific Problem Solving

Karlsruhe
September 23 – 27, 1991



Co-chairs of the Program Committee:	Bo Einarsson, Lloyd D. Fosdick
Co-editors:	Patrick Gaffney, Elias Houstis
Local Organization:	Ulrich Kulisch, Adolf Schreiner

The meeting is co-organized by the Institute for Applied Mathematics of Karlsruhe University.

Edited by: Gerd Bohlender, Universität Karlsruhe, September 9, 1991

Front cover: symbols of IFIP and Karlsruhe University,
view of Karlsruhe in 1739 (engraving by Christian Thran)

Printed and bound by: SCHNELLDRUCK GRÄSSER
7500 Karlsruhe 1, Humboldtstraße 1, Tel. 0721/615050

Scientific Program, Registration and Travel Information

as of September 9, 1991

Table of Contents

IFIP

International Federation for Information Processing

IFIP WG 2.5 Working Conference 6

Programming Environments for High Level Scientific Problem Solving

Karlsruhe, September 23 – 27, 1991

Program committee: Michel Bercovier, Jacques Calmet, Ifay Chang, Bo Einarsson, Stuart Feldman, Brian Ford, Lloyd Fosdick, Pat Gaffney, Morven Gentleman, Elias Houstis, Ulrich Kulisch, John Rice, Adolf Schreiner, Hans Stetter, Claus Unger, Mladen Vouk.

The meeting is co-organized by the Institute for Applied Mathematics of Karlsruhe University.

Co-chairs of the Program Committee:	Bo Einarsson, Lloyd D. Fosdick
Co-editors:	Patrick Gaffney, Elias Houstis
Local Organization:	Ulrich Kulisch, Adolf Schreiner

Opening of Conference:	Monday, September 23, 9:30
Closing of Conference:	Friday, September 27, 12:30
Conference auditorium:	NTI in Nachrichtentechnisches Institut, corner of Engesserstrasse and Zirkel

Acknowledgements

The organizers would like to express their gratitude to the following organizations for supporting the conference:

the scientific program of the conference is supported by a grant from IBM Germany,
the participation of American scientists is supported by a grant from the European Research Office of the U.S. Army,
two workstations for demonstrations in the auditorium are provided by Sun Microsystems,
a room for the conference secretariate on Sunday is provided by Hotel Kaiserhof.

Scientific Program

(as of September 5, 1991)

Sessions

For each session the responsible / assistant are given. They will also be session chairman / discussant.

Monday, September 23
9:30 - 12:30 **1. Introduction**
Lloyd Fosdick / Mladen Vouk
14:00 - 18:00 **2. Intelligent Assistants**
John Rice / Patrick Gaffney

Tuesday, September 24
9:00 - 12:30 **3. Mathematical Methods**
Jacques Calmet / Hans Stetter
14:00 - 18:00 **4. Systems and Tools**
Mladen Vouk / [Stu Feldman] / Theodorus Dekker

Wednesday, September 25
9:00 - 12:30 **5. System Engineering I**
Stu Feldman / [Claus Unger]

Thursday, September 26
9:00 - 12:30 **6. Interaction and Visualization**
Morven Gentleman / [Ifay Chang]
14:00 - 21:30 **7. Open Session**
Elias Houstis / Bo Einarsson

Friday, September 27
9:00 - 12:30 **8. System Engineering II**
Michel Bercovier / Brian Ford

Monday, September 23, Morning Session

1 Introduction

(Lloyd Fosdick / Mladen Vouk)

09:30	Bo Einarsson	S	Opening
	Ulrich Kulisch	D	Welcome
	Lloyd D Fosdick	USA	Program
09:45	Siu Shing Tong	USA	**Key Note:** Integration of Symbolic and Numerical Methods for Optimizing Complex Engineering Systems
10:30	Discussion		
10:40	Break		
11:10	W M Gentleman	CDN	Symbiotic Computation: Opportunities and Complications
11:35	Discussion		
11:40	Paul C Abbott	GB	Problem Solving Using Mathematica
12:05	Discussion		
12:15	Session Discussion		
12:30	Lunch		

Monday, September 23, Afternoon Session

2 Intelligent Assistants

(John Rice / Patrick Gaffney)

Dynamic selection of algorithms.
Use of knowledge bases for problem solving.
Accuracy control and estimation, self-validating systems.

14:00	Siegfried Rump	D	Accuracy Control and Estimation, Self-Validating Systems and Software Environments for Scientific Computation
14:30	Discussion		
14:40	Willi Schoenauer	D	Polyalgorithms with Automatic Method Selection for the Iterative Solution of Linear Equations and Eigenproblems
15:10	Discussion		
15:20	Break		
15:50	John A Nelder	GB	The Computer as Statistical Assistant
16:20	Discussion		
16:30	Donald F Boisvert	USA	Toward an Intelligent System for Mathematical Software Selection
17:00	Discussion		
17:10	Session Discussion		
17:20	Poster Session		

Tuesday, September 24, Morning Session

3 Mathematical Methods

(Jacques Calmet / Hans Stetter)

Graphical, symbolic and numerical methods.

09:00	Masaaki Shimasaki	J	Fractals in Quaternions and their Application to Computer Graphics
09:30	Discussion		
09:40	Vladimir P Gerdt	SU	Computer Algebra Tools for Higher Symmetry Analysis of Nonlinear Evolution Equations
10:10	Discussion		
10:20	Break		
10:50	Bruno Buchberger	A	Groebner Bases in Mathematica: Enthusiasm and Frustration
11:20	Discussion		
11:30	Dominique Duval	F	Examples of Problem Solving using Computer Algebra
12:00	Discussion		
12:10	Session Discussion		
12:30	Lunch		

Tuesday, September 24, Afternoon Session

4 Systems and Tools

(Mladen Vouk / [Stu Feldman] / Theodorus Dekker)

Mixed language programming.
Declarative, dynamic and visual programming systems.
Tools for integration and portability.
Tools for performance measurement and evaluation.

14:00	Stuart Feldman	USA	Environments for Large-Scale Scientific Computation
14:30	Discussion		
14:40	Hans Zima	A	Software Tools for Parallel Program Development
15:10	Discussion		
15:20	Break		
15:50	Wayne R Dyksen	USA	Fortran Interface Blocks as an Interface Description Language for Remote Procedure Call
16:20	Discussion		
16:30	(Nobutoshi Sagawa) Neil Hurley	IRL	An Integrated Problem Solving Environment for Numerical Simulation of Engineering Problems
17:00	Discussion		
17:10	Session Discussion		
17:20	Demonstrations		

Reception

19:00	Reception in the Casino of the Constitutional Court

Wednesday, September 25, Morning Session

5 System Engineering I

(Stu Feldman / [Claus Unger])

Integration of numerical, symbolic and graphical methods.
Integration of libraries and multiple problem solving systems.
Efficient utilization of computing resources, parallel and distributed architectures, graphics engines.

09:00	Michael Clarkson	CDN	Expert System as an Intelligent User Interface for Symbolic Algebra
09:30	Discussion		
09:40	Alfonso Miola	I	Design and Implementation of Symbolic Computation Systems
10:10	Discussion		
10:20	Break		
10:50	Elias Houstis	USA	Parallel Ellpack
11:20	Discussion		
11:30	Jim Purtilo	USA	Dynamic Software Reconfiguration Supports Scientific Problem Solving Activities
12:00	Discussion		
12:10	Session Discussion		
12:30	Lunch		

Excursion

14:00 Excursion and Banquet in the Baden-Baden area

Thursday, September 26, Morning Session

6 Interaction and Visualization

(Morven Gentleman / [Ifay Chang])

Techniques for problem description.
Graphical presentation of results.
Multiple modes of I/O (digital, analog, graphical, audio,...).
User interaction and feedback.

09:00	Yukio Umetani	J	Visual DEQSOL: A Visual and Interactive Environment for Numerical Simulation
09:30	Discussion		
09:40	Eric Grosse	USA	Display of Functions of Three Space Variables and Time Using Shaded Polygons and Sound
10:10	Discussion		
10:20	Break		
10:50	(Frans C A Groen) Hans J W Spoelder	NL	Distributed Visual Programming Environment: an Attempt to Integrate Third Generation Languages with Advanced User Environments
11:20	Discussion		
11:30	Peter M Dew	GB	Visualization and its Use in Scientific Computation
12:00	Discussion		
12:10	Session Discussion		
12:30	Lunch		

Thursday, September 26, Afternoon/Evening Session

7 Open Session

(Elias Houstis / Bo Einarsson)

14:00-17:30			Afternoon Session
	Johan A van Hulzen	NL	Automated Generation of Optimized Numerical Code for Jacobians and Hessians
	Niklas Holsti	SF	Transcript Editing, A Simple User Interface Tool
	Allan Bonadio	USA	Mathematical User Interfaces for Graphical Workstations

Additional papers to be announced later.

19:00-21:30

Evening Session.
To be announced later.

Friday, September 27, Morning Session

8 System Engineering II

(Michel Bercovier / Brian Ford)

09:00	Kevin Broughan	NZ	SENAC: Lisp as a Platform for Constructing a Problem Solving Environment
09:30	Discussion		
09:40	I A Nicolayev	SU	Complex Application of Graphical, Symbolic and Numerical Methods in Packages for Solving Mathematical Modeling Problems
10:10	Discussion		
10:20	Break		
10:50	Colin W Cryer	D	The ESPRIT Project FOCUS
11:20	Discussion		
11:30	Steve J Hague	GB	Use of Knowledge Bases for Problem Solving – The FOCUS Approach
12:00	Discussion		
12:10	Session Discussion		
12:20	Final Discussion		
12:30	Close		

Travel Information

IFIP WG 2.5 Conference **Programming Environments for High Level Scientific Problem Solving** will be held at Karlsruhe University. Karlsruhe is a town with 280 000 inhabitants in the south-west of Germany. It is situated in the valley of the river Rhine, about 60 kilometers south of Heidelberg and 60 kilometers north-east of Strasbourg (France). In Karlsruhe, Baron von Drais invented the wooden bicycle and Heinrich Hertz discovered the electro-magnetic waves. Today, Karlsruhe is the seat of the Federal Supreme and Constitutional Courts.

The conference will begin Monday morning, September 23, and will end Friday noon, September 27. Registration will be held Sunday, September 22 and Monday morning, September 23.

Arrival by Plane and Train

Karlsruhe can be easily reached by public transport since intercity trains from Frankfurt to Basel (Switzerland) arrive at Karlsruhe every hour. The easiest way to get to Karlsruhe from overseas is:

- Fly to Frankfurt; from the baggage claim you can proceed to the underground railway station
- Take local train "S-Bahn" to Frankfurt central station ("Hauptbahnhof", "Hbf"). Alternatively take "S-Bahn" to Mainz. **Important:** do not enter **S-Bahn** without a ticket, no tickets are sold on these trains!
- From Frankfurt or Mainz, take intercity/eurocity train to Karlsruhe via Mannheim

Remarks: For most trains from Frankfurt central station to Karlsruhe, you have to change in Mannheim. For some trains you have to pay an increased "intercity express" fare.

Timetable: The trip from Frankfurt airport to Karlsruhe takes approximately 90 minutes; trains leave every hour until about 22:00.

From Frankfurt to Karlsruhe: the trip from the airport to Frankfurt central station takes 10 minutes. From there intercities usually leave at xx:46 and arrive in Karlsruhe 72 minutes later at xx+1:58. The regular timetable starts at 7:46 and ends at 21:46 with some gaps (7:46, 10:46,...,19:46 not on Sundays, 19:46, 20:46 not on Saturdays, etc.); there are additional trains which do not fit in the scheme.

From Mainz to Karlsruhe: the trip from the airport to Mainz takes 24 minutes. From there intercities leave usually at xx:44 and arrive in Karlsruhe 74 minutes later at xx+1:58. The regular timetable starts at 7:44 and ends at 22:44 with some gaps (e.g. 20:44 and 22:44 not on Saturdays); there are additional trains which do not fit in the scheme.

From Karlsruhe to Frankfurt airport: most intercities leave Karlsruhe at xx:59. The regular timetable starts at 5:59 and ends at 20:59 with some gaps (5:59 and 7:59 not on Sundays, 9:59 not on Saturdays and Sundays). Usually, you can stay on the train till Mainz or change at Mannheim to the train to Frankfurt central station. From Mainz or Frankfurt, take S-Bahn to the airport. Via Frankfurt the trip takes about 95 minutes, via Mainz about 115 minutes. There are additional trains which do not fit in the scheme.

Railway tickets can be bought at Frankfurt airport or they can be ordered in advance at a reduced rate by using the KTS ticket ordering form (only in connection with a room reservation). Please, send this form and your payment directly to **Karlsruher Tagungs- und Touristikservice**, as indicated on the form; do not send the form to the conference address, do not include that payment in your registration fee.

Alternative airports are Stuttgart, Basel (Switzerland), or Strasbourg (France).

Location of the Conference Auditorium

The campus of Karlsruhe University is north-east of the town center, about 10 minutes by foot. The conference auditorium NTI of "Nachrichtentechnisches Institut" is situated on the campus at the corner of **Engesserstrasse** and **Neuer Zirkel** (marked $\boxed{\text{u}}$ in building 30.10 on campus map). It is a five minutes' walk north of tramway station **Berliner Platz / Universität** and about seven minutes from **Marktplatz**.

Public Transport in Karlsruhe

Most places of interest (University campus, town center) are within a walking distance. In addition, Karlsruhe has an efficient public transport system: tramways on the main lines, connecting busses on the less important lines.

Regular fares (as of September 1, 1991) for tramways and busses are DM 2.50 for a one way ticket, DM 7.00 for 4 trips, and DM 6.00 for a 24 hour ticket (any number of trips within 24 hours, valid for 2 adults plus 2 children). Special tickets can be bought from **Karlsruher Tagungs- und Touristikservice** at the reduced rate of DM 2.50 per day per person by using the KTS ordering form (Note: this reduced rate is only available in connection with a room reservation, see the remarks for railway tickets above).

Note: When first entering a tramway or bus in Karlsruhe, you have to insert your ticket into the yellow box in order to stamp the time on it. Your ticket is **invalid** without a time stamp.

Parking in Karlsruhe

The campus of Karlsruhe University can only be entered by car with a special permit, which can be obtained in the conference office. **Note:** when parking on the campus, please display this permit in your car.

In addition, there is a big free parking on Adenauerring, on the north-east of the campus. From there you can walk to the auditorium in about 10 to 15 minutes. There are several car parks in the west and in the south of the campus. Prices are about DM 1.50 per hour or DM 13.5C per day; car parks usually are closed some hours during the night.

Hotel Reservation

Important: There are other meetings in Karlsruhe in the same week with several hundred participants; **please make your hotel reservation as soon as possible!**

An accomodation guide with a list of hotels in Karlsruhe and a town map are enclosed; please note: prices in this guide are of 1989 and have increased in the meantime by 10 to 20 percent. Please, make a hotel reservation (and, if desired, a ticket reservation) by sending the KTS hotel reservation form directly to **Karlsruher Tagungs- und Touristikservice**. You can indicate special wishes (e.g. location of the hotel) on your room reservation card.

Hotel reservation is free of charge, no deposit is required. If you order railway or tramway tickets with the KTS reservation form, please pay by attaching a check in Deutschmark drawn on a German bank; do not send this form to the conference address, do not include that payment in your registration fee.

Alternatively, you can select a hotel from the accomodation guide as well and make a reservation directly with the hotel. Kaiserhof and Kübler, for instance, are nice hotels nearby. The area code for phone calls from foreign countries is +49 721. For several hotels the fax number is included in the list as well. For mail to Karlsruhe, please add the zip code D-7500 in front of the town. For **Karlsruher Tagungs- und Touristikservice** the full address is

Address: KTS
Karlsruher Tagungs- und
Touristikservice
Verkehrsverein Karlsruhe e.V.
Bahnhofplatz 6
D - 7500 Karlsruhe 1
Germany
Phone: +49 721 3553 0
Fax: +49 721 3553 43

Meals

In most hotels, breakfast buffet is included in the hotel rate. Lunch can be taken at several places:

- The **Mensa and Cafeteria** of Karlsruhe University are on the east of the campus, about a 5 - 10 minutes' walk from the auditorium. Prices for guests are DM 7.70 for a menu (including soup and dessert). A choice of several varying menus is offered on Monday – Friday. Small dishes are available in the self service cafeteria starting at about DM 3.00.

- There are dozens of small restaurants on Kaiserstrasse and its side roads within about 10 minutes from the auditorium. A map of restaurants will be available in the conference office.

- A limited number of tables will be reserved for conference participants in **Gastdozentenhaus** which is situated on campus in the building next to the conference auditorium. These tables are reserved for speakers and session chairmen / discussants who are invited for lunch on the day before their session. Monday speakers, session chairmen / discussants are invited on Friday.

Additional Information

Summer 1991 was dry and hot in Karlsruhe (the local newspaper said it was the driest August since 1876). The **weather** in the end of September in Karlsruhe is frequently still warm and friendly. Street cafés are still open in the pedestrian zone. However, the evenings may be cool and there may be some rain. Summertime (**daylight savings time**) ends on Saturday night, September 28.

Credit cards are by far not as common in Germany as in e.g. France or the United States. Most smaller shops and restaurants accept only cash or eurocheque, no foreign currency. You should exchange some money, e.g. at Frankfurt airport.

English is more or less understood by many Germans. But apart from airports, there are nearly no signs or other information in English.

Opening hours of shops and banks are regulated by law. Only shops in railway stations, airports, and gas stations are allowed to be open after regular shopping hours

open on ...	Mon–Wed&Fri	Thu	Sat	Sun
shops	9:00 - 18:30	9:00 - 20:30	9:00 - 14:00	closed
banks	8:30 - 16:00	8:30 - 18:00	closed	closed
at Karlsruhe central station:				
tourist info.	8:00 - 19:00	8:00 - 19:00	8:00 - 13:00	closed
shops	7:30 - 20:30	7:30 - 20:30	7:30 - 20:30	7:30 - 20:30
bank	7:00 - 20:00	7:00 - 20:00	7:00 - 20:00	9:00 - 13:00

Social Program

The following social events are envisaged. Times of departure are, however, still subject to change. Please indicate your interest in particular events on your registration form.

Sunday, September 22, 12:00 – 21:00

Welcome in the Hotel Kaiserhof, Am Marktplatz. The conference office will be located there. Dining facilities are available.
(Not included in registration fee)

Monday, September 23, 20:00

Concert of **Virtuosi Saxoniae** with Ludwig Güttler (trumpet) in the Town Hall, Brahms-Saal.
(Not included in registration fee; the price is DM 46.00)

Tuesday, September 24, 19:00

Get together party with wine, cheese and salad bar at the Casino of the Constitutional Court, close to Karlsruhe castle.
(Included in registration fee for participants and accompanying persons)

Wednesday, September 25, 14:00 – 23:00

Conference banquet at Baden-Baden (about 30 minutes by bus from Karlsruhe). Busses will leave from Karlsruhe at 14:00; in Baden–Baden you can join a guided city walk and go to the top of mountain Merkur by cable car. In the evening at 19:00, Banquet in Hotel Plättig in Black Forest near Baden–Baden. Busses will leave back to Karlsruhe at 21:00 and 23:00, so participants have an opportunity to revisit Baden–Baden after the banquet (bus trip and banquet: included in registration fee for participants and accompanying persons).
Meeting point: near NTI auditorium.

Guided tours of the Computing Center of Karlsruhe University will be arranged during the conference.

In addition to these events, several excursions will be offered for accompanying persons. These trips have to be paid for individually; please do not include the payment in your registration fee. However, please help us in planning these excursions by informing us as soon as possible about the estimated number of participants.

The following excursions are planned (leaflets with detailed information are available in the conference office):

- **Monday, September 23, 11:00 – 12:30**
  After the opening of the conference, all accompanying persons are invited by Ursula Kulisch for Morning Coffee in Gastdozentenhaus (the building next to the auditorium).

- **Monday, September 23, 14:15 – 16:30**
  Walking tour to **Staatliche Majolikamanufaktur**, manufacture of pottery, statues, decorated tiles. (Guided tour of the factory is about DM 3.00 – DM 6.00 depending on the number of participants).
  Meeting point: terrace on north side of Karlsruhe Castle (Schloßcafé).
  An exhibition of Russian and Soviet china from St. Petersburg / Leningrad is opened on Tuesday, September 24 in Badisches Landesmuseum in Karlsruhe Castle.

- **Tuesday, September 24, 10:00 – 15:00**
  Trip by tramway to **Bad Herrenalb**, a small spa in the northern part of Black Forest (tramway ticket DM 6.00).
  Meeting point: tramway station near pyramid on Marktplatz.

- **Wednesday, September 25, 10:00 – 11:30**
  Walking tour to the Art Gallery **Staatliche Kunsthalle**. (Admittance is free; guided tour: DM 5.00)
  Meeting point: 9:45 at pyramid on Marktplatz or 10:00 at main entrance of Kunsthalle, Hans-Thoma-Strasse.

- **Thursday, September 26, 12:30 – 19:00**
  Tour by train to Pforzheim which is famous for its manufacture of jewelry; visit of **Museums for Jewelry and Manufacture of Jewelry**. (Trip approximately 30 kilometers by train, price about DM 20.00 including railway ticket, admittance fee and guided tour of the museum)
  Meeting point: tramway station near pyramid on Marktplatz.

The trips to Heidelberg and Deidesheim unfortunately had to be cancelled. However, we will give you information about Heidelberg so that you can easily make the trip on your own, for example on Friday. There is a direct railway line (57 kilometres, 30 – 60 minutes by train).

General Information

Conference Address (Local Arrangements)

Address: IFIP WG 2.5 Conference 1991
Prof. Ulrich Kulisch
Institut für Angewandte Mathematik
Universität Karlsruhe
Kaiserstrasse 12
D-7500 Karlsruhe
Germany
Phone: +49 721 608 2680 (secretariate in institute)
+49 721 608 2839 (Dr. Gerd Bohlender)
Conference phone: +49 721 608 2696 (conference secretariate at
NTI auditorium, Monday, Sept. 23 – Friday, Sept. 27)
Fax: +49 721 695283
Email: AE15@DKAUNI2.BITNET (Dr. Gerd Bohlender)
(Email, X.400: ae15@ibm3090.rz.uni-karlsruhe.dbp.de)

Technical Organization (Computers, Demonstrations, etc.)

Address: IFIP WG 2.5 Conference 1991
Prof. Adolf Schreiner
Rechenzentrum
Universität Karlsruhe
Kaiserstrasse 12
D-7500 Karlsruhe
Germany
Phone: +49 721 608 3754 (secretariate)
+49 721 608 2068 (Dipl.-Ing. Reinhard Strebler)
Fax: +49 721 32550
Email: RZ06@DKAUNI2.BITNET (Prof. Schreiner)
RZ40@DKAUNI2.BITNET (Dipl.-Ing. R. Strebler)

Registration

Please use the enclosed registration form and send it by air mail to the conference address. The registration fee is DM 450 for participants and DM 120 for accompanying persons. Registrations are supposed to arrive before June 30, 1991. Later registrations will be charged with additional DM 50.00 per person. Please pay the appropriate amount in Deutschmark (abbreviated DM in Germany, or DEM in international usage)

- by sending a **bankers check** in Deutschmark, drawn on any German bank to the conference address (please indicate on your check "**IFIP WG 2.5 Conference**"), or
- by remittance on the account of "Karlsruher Hochschulvereinigung", account number 400 242 91 00, Baden-Württembergische Bank, Karlsruhe, SWIFT-Code BW BK DE 6K 660 (in Germany: Bankleitzahl 660 200 20). Please indicate on your remittance "**IFIP WG 2.5 Conference**".

All payments have to be free of charges to the local organizer. We are sorry that we cannot accept credit cards or personal checks.

Cancellation of Registrations: Cancellations in writing will be honored, less 20% handling charge, prior to September 1, 1991. After this date, partial refunds will be made at the discretion of the conference organizers.

Conference Office

On Sunday, September 22, 12:00 – 21:00, the conference office will be located in Hotel Kaiserhof, Marktplatz, phone +49 721 26615.

During the conference, the conference office will be located in front of auditorium NTI (marked $\boxed{u}$ in building 30.10 on the campus map). On Monday morning, September 23, the office will open at 8:30.

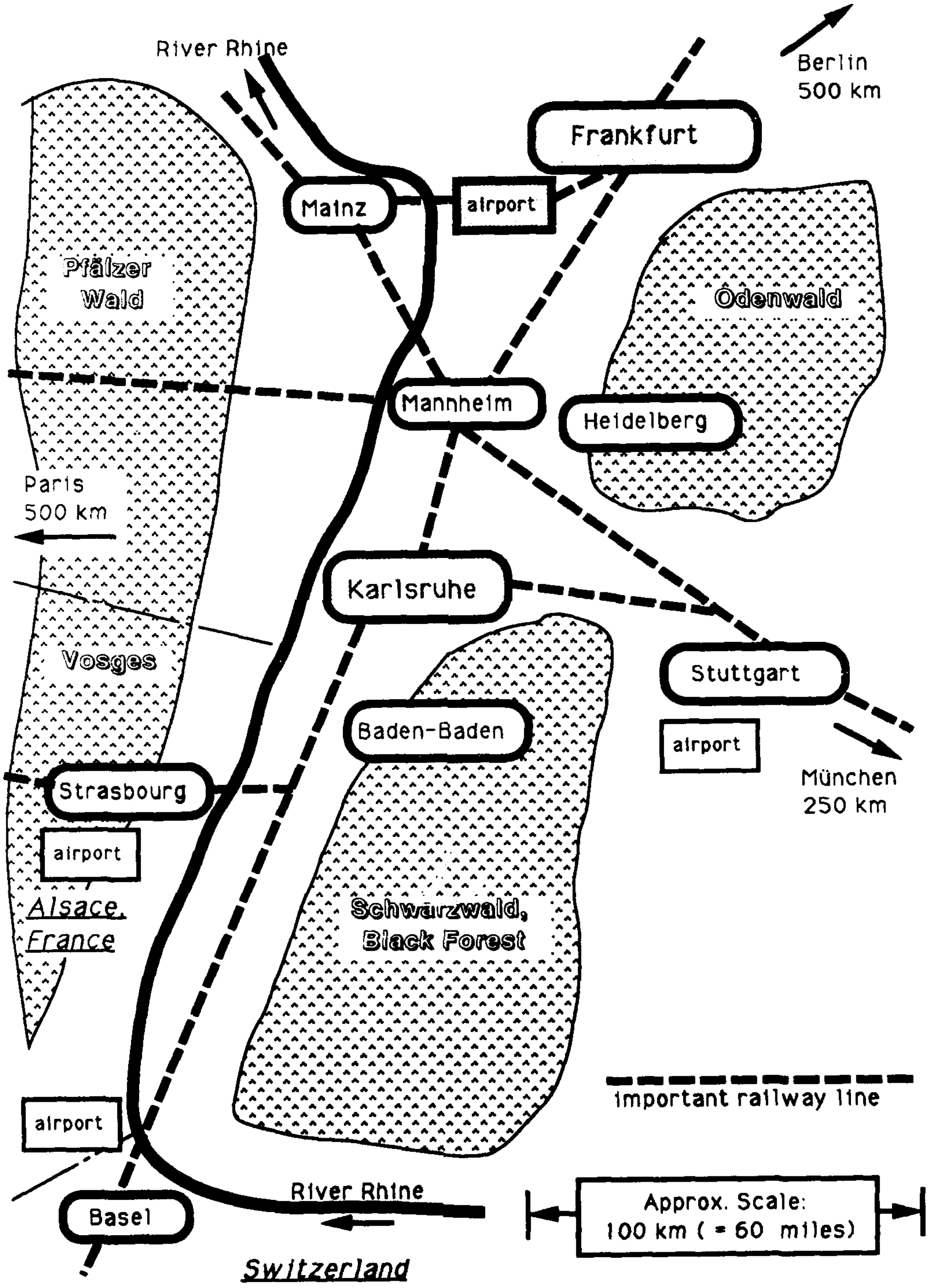
The phone number of the conference office is +49 721 608 2696; email is available in the conference office; fax (number +49 721 695283) is available in the Mathematics building, 100 metres from the auditorium.

Badges

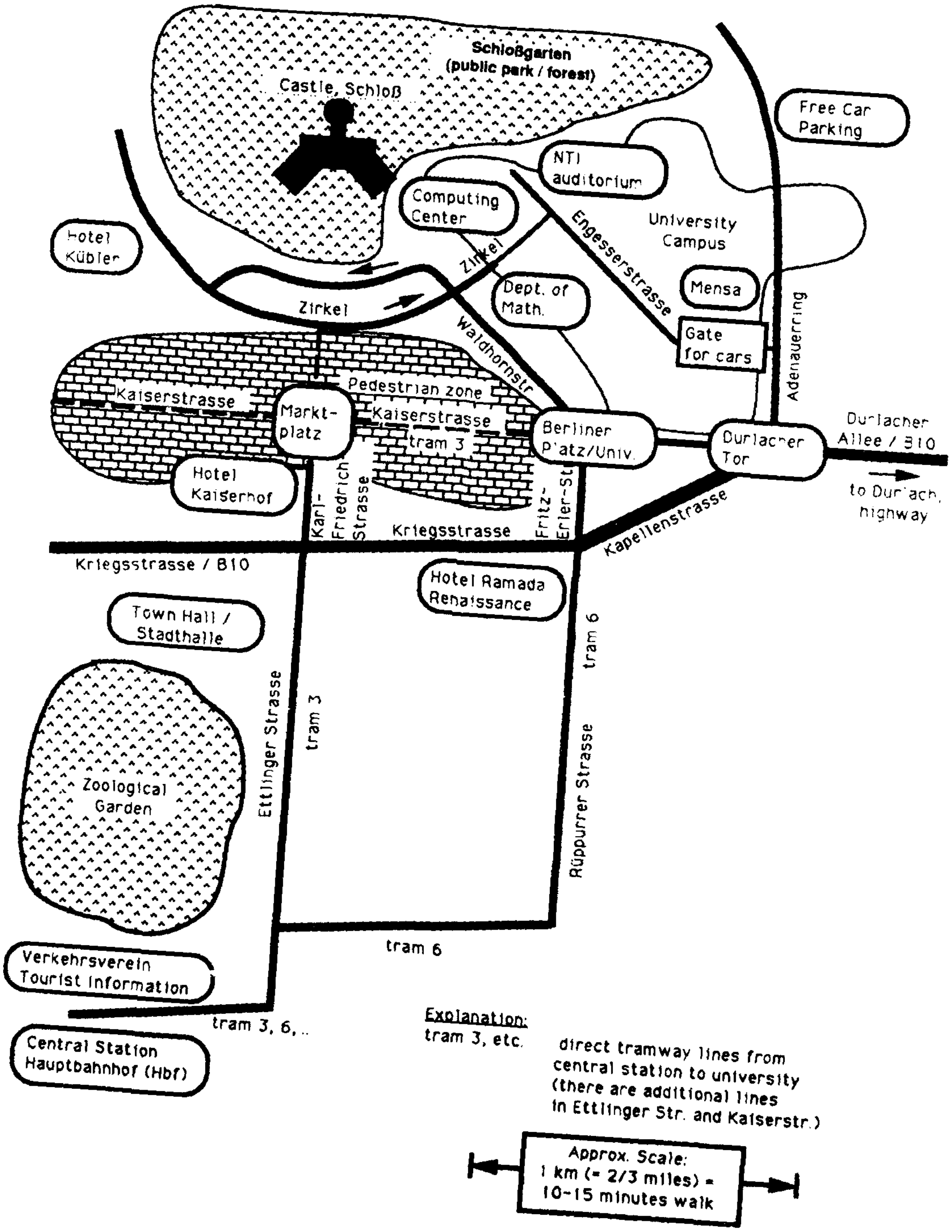
white badges:	conference participants
... with IFIP symbol:	... WG 2.5 members
yellow badges:	local persons who assist in organizing the conference
pink badges:	accompanying persons

Accompanying persons can use their badges during social events, if they like.

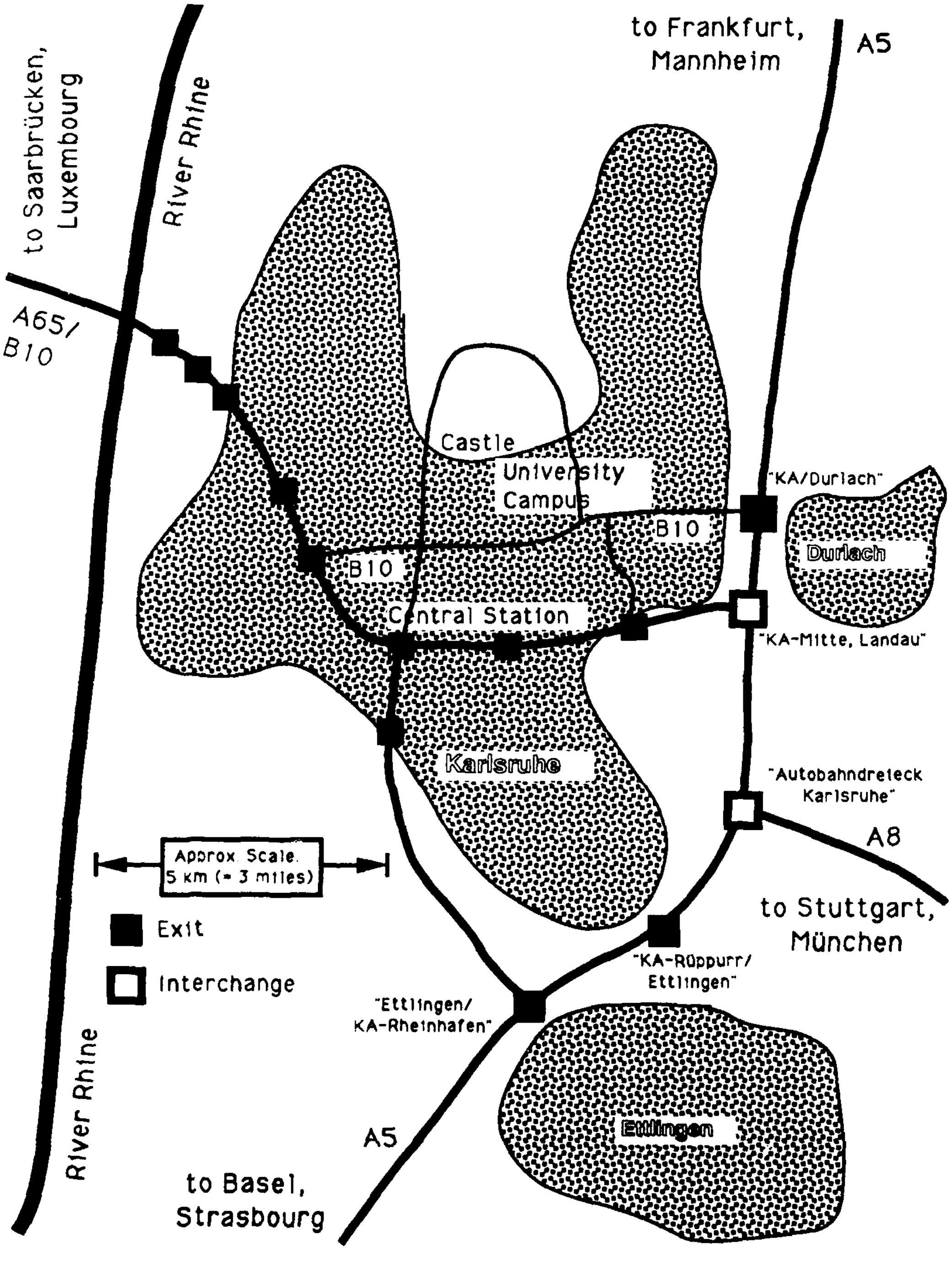
Railway Map of Rhine Valley



Karlsruhe: Town and University



Important Roads in the Karlsruhe Area



DAJA45 91 M0319
R&D 6809-EE-02

AD-A243 783

IFIP

International Federation for Information Processing

Working Conference on Programming Environments for High–Level Scientific Problem Solving

23 – 27 September 1991, Karlsruhe, Germany

Prepared by Bergen Scientific Centre

IBM

Integration of Symbolic and Numerical Methods for Optimizing Complex Engineering Systems

Siu Shing Tong
Corporate Research and Development
General Electric Company
PO Box 8
Schenectady, NY 12301

A new software system called Engineous combines symbolic and numerical methods for the design and optimization of complex engineering systems. Engineous combines advanced computational techniques — genetic algorithms, expert systems, and object-oriented programming — with conventional methods — numerical optimization and simulated annealing, to create a design optimization environment that can be applied to computational models in various disciplines. Engineous has produced designs with higher predicted performance gains than current manual design processes, on average a 10-to-1 reduction of turnaround time, and has yielded new insights into product design. It has been applied to the design of an aircraft engine turbine, molecular electronic structure, cooling fan, DC motor, electrical power supply, nuclear fuel rod, and the concurrent aerodynamic and mechanical, preliminary and detailed design of 3D turbine blades.

Introduction

In today's dynamic market environment, industry faces the challenge of timely development of new products to meet market needs. New, complex computational codes that take advantage of increasingly faster and cheaper computing power are being used routinely for new product design. One obstacle to producing good new designs quickly, however, is the scarcity of engineers who combine an understanding of product design with an ability to rapidly learn how to use these new codes. At present, the few who do combine these skills must still go through the tedious process of manually iterating analysis codes to obtain optimum design. An additional obstacle results from the need for a large group of engineers from various disciplines, with diverse expertise, to work together. The difficulty of communication between these engineers has created a bottleneck in developing new products. Partly because of these obstacles, it often takes years to bring a product from concept to market. Automating product development to reduce development cycle time is now a priority task for many industrial organizations.

Numerical and Symbolic Optimization Methods

The use of computers to search for an optimum configuration for a complex system is, of course, nothing new. Numerous optimization techniques have been developed and demonstrated successfully on many relatively simple problems with few parameters and simple analytical models. These techniques can be categorized as numerical or symbolic.

Numerical Methods

Numerical methods treat all design variables as independent numbers without regard to their characteristics and significance, often making use of mathematical principles such as gradients to guide the search. Examples of numerical methods include numerical optimization, simulated annealing, and inverse method.

Numerical optimization. A substantial amount of work has been done on numerical optimization over the past twenty years. For nonlinear constrained optimization, the techniques of sequential linear programming, sequential quadratic programming, modified method of feasible directions, and the generalized reduced gradient method have emerged as popular numerical optimization techniques [Gabriele, 1988; Vanderplaats 1984, 1988]. These optimization techniques use local numerical gradient approximations to determine how to move from one trial solution to another in order to obtain maximum gain of the objective function. They have been successfully applied to optimization problems in many domains [Vanderplaats, 1984]. The advantages of numerical optimization are its mathematical underpinnings, general applicability to engineering designs, and wide application base. The disadvantages are its inability to exploit domain knowledge, extreme sensitivity to both problem formulation [Gero, 1988] and algorithm selection, the large amount of computational effort required, a tendency to be trapped in local optima, and the assumptions of both design variable independence and continuity within the parameter space.

Simulated annealing [Kirkpatrick, Gelatt, and Vecchi, 1981; Ban den Bout]. This method got its name from the analogy of growing crystal by gradually lowering the energy level to allow the element to align itself through random motions. The controlled random process helps avoid the problem of being trapped in a local optimum in a hill-climbing procedure. This method often requires many iterations.

Inverse method. For some special problems it is possible to invert the governing equations so that the configuration satisfying certain design objectives is the solution of the computational code. Some examples are shown in Giles, Drela, and Thompkins [1985] and Tong [1984].

Symbolic Methods

Replacing a total reliance on mathematics, symbolic methods emulate human approaches and attempt to make use of accumulated knowledge — either explicitly input by humans or generated from the iteration process — to reduce the number of iterations required. The characteristics and significance of various parameters are often retained and explored. Examples of symbolic methods include expert systems, heuristic search, and genetic algorithms.

Expert systems. Expert systems codify domain-specific knowledge in the form of IF THEN rules. Expert systems have been used in designs in many domains, including refrigerator fans [Tong, 1986], VLSI circuits [Jabri, 1987], and bridges [Adeli, 1986]. The main advantages of using expert systems are the capabilities of using engineers' experience to obtain good solutions rapidly and to trace the rationale that was used in the process. However, the applicability of each system is generally restricted to a very narrow domain, and the development effort can be prohibitively expensive for complex problems. Often, attempts to develop such systems fail because of the inability of humans to express all of their knowledge and because of the errors made in the transfer process from human to computer [Zhou, 1987].

Heuristic search [Winston, 1984]. Heuristic search techniques keep a record of the past solution path and look ahead at possible future moves, so that iterations can be backtracked if problems arise and planned ahead to minimize bad moves. They rely on common sense logic but generally have the same problems as numerical optimization such as dealing with local optima.

Genetic algorithms. Genetic algorithms take an initial set, or population, of design points and manipulate that set with the genetic operators of selection, crossover, and mutation to arrive at an optimal design. These general purpose search strategies attempt to strike a balance between exploration and exploitation of a parameter space [Holland, 1975]. They are based on the heuristic assumptions that the best solutions will be found in regions of the parameter space containing relatively high proportions of good solutions and that these regions can be explored by genetic operators. The advantages of genetic algorithms are that they search from a set of designs and not from a single design, they are not derivative based, they work with discrete and continuous parameters, and they explore and exploit the parameter space [Goldberg, 1989]. The major disadvantage is the excessive number of runs of the design code required for convergence. Another disadvantage is that once a genetic algorithm has found a good solution it does not continue exploring around that solution to find a better one.

Optimizing Complex Engineering Systems

Although optimization techniques are used widely on relatively simple problems, they have had only limited application on complex engineering systems involving large-scale analytical codes. Problems in this class have one or more of the following characteristics:

- Large numbers of parameters with mixed types (e.g., real, discrete, symbolic, vector)
- Analysis codes that are complex, CPU-intensive, difficult to use, and continuously evolving
- Design parameters that have complicated relationships and cannot be manipulated independently
- Design space with multiple optima
- Design concerns that cannot be modeled analytically
- Multiple conflicting goals where the most advantageous trade-off is not obvious
- Usefulness of past experience and understanding of the physics of the problem in helping the designer to locate a good solution quickly

Factors Limiting the Use of Optimization Tools

Because of a common misconception that the primary role of a designer is to obtain a design that is an "optimum" based on a numerical measure of merit, many computational design tools were written in a "black-box" manner. In reality, understanding the characteristics of the design space and behavior of the proposed designs is as important to a designer as obtaining the numerical optimum. In many problems, the operating conditions, the choice of variables, the constraint boundaries, and the weighting factor of various objectives are by no means exact. Information needs to be extracted beyond obtaining an "optimum" design based on a static condition.

Even in an ideal situation where a numerically optimized solution is sought, the large number of parameters plus the slow turnaround time of the analysis codes make it time-consuming to search through the design space numerically without utilizing knowledge of the domain.

The complexity of some of the optimization methods adds to the problem. Systems that require frequent tuning of search parameters by users are merely transferring the iteration of physical parameters to the iteration of numeric parameters unfamiliar to the designers. Installation of new applications is often too labor-intensive, particularly for new technology areas where analysis codes continue to evolve.

Integration of Symbolic and Numerical Methods: Engineous Shell

The Engineous project was initiated at the GE Corporate Research and Development Center in 1984 to develop a new approach to the design optimization of complex systems. It was decided that an integrated environment with various facilities addressing the needs of complex engineering system designs would be appropriate.

The basic approaches of Engineous are:

- Unified automation, optimization, and integration: A single system that has the capability to automate the design process by emulating designer engineers, to explore the design space by numerical and symbolic optimization with or without any knowledge, and to combine multidisciplinary design knowledge.
- Generic shell: Problem-specific information is separated from generic design functions and is organized as knowledge bases.
- Integration of numerical and symbolic optimization technologies: To accommodate a wide range of engineering problems having various combinations of continuous, discrete, and symbolic parameters, different amount of design knowledge, and drastic differences in the shape of the objective functions, a hybrid approach is provided.
- Rapid coupling to computational analysis codes without modification of the code.

The following description details the design automation, optimization, and integration approaches used by Engineous.

Design Automation

The ability to manage the execution of a series of complex analysis codes without modification is an essential feature. The capabilities needed include:

- Preparation of input files and extraction of relevant output parameters
- Execution of analysis code and monitoring of its progress (e.g., abort if infinite loop)
- Management of the execution sequence
- Management of the data flow from one program to another
- Capture of design procedures (e.g., which parameters are allowed to vary given certain conditions?)
- Capture of design parameter properties (e.g., what combinations of design parameters are feasible? what are the significant digits?)

Engineous provides generic functionality to perform these tasks. Problem-specific information for each application is captured in a Design Knowledge Base (KB) within Engineous. Each application will have its own Design KB. The Design KB can be further decomposed logically into: Design Parameter KB, Program KB, Program Sequence KB, and Design Rules KB.

Design Parameter KB. This KB classifies each parameter as belonging to one or more of the 12 default types and attaches various properties to it. For example, Engineous attaches current value, minimum increment, default value, etc., to a "design" parameter. The "material" parameter is a symbolic type that can only take on a predefined set of symbolic values. The tip-speed, rpm, and tip-diameter parameters are coupled parameters, whose relationship is stored; any change in one parameter affects the others.

Program KB. This KB captures information on: (i) how to set up the input files, (ii) how to execute an analysis code, (iii) what the code's input and output parameters are, and (iv) how to extract useful information from analysis output files for Engineous. The Program KB also manages all pre- and post-processing programs that need to be executed for each analysis code.

Program Sequence KB. This KB manages the execution sequence. It does not contain a hardwired procedure but rather retains only the allowable transition between programs, i.e., for each program, what programs besides itself can follow it. Which program actually runs is determined at run time depending on the parameters to be modified.

Design Rules KB. This KB captures human design knowledge in the form of logical steps a designer would go through. Engineous uses a special rule syntax developed to represent analysis-code-based iterative design knowledge. This knowledge representation is based on the observation that only a limited set of goals, premises, and actions are used during iterative design. An Engineous design rule might be "translated" into English as follows:

```
      Goal:  To increase X
Conditions:  Y > 3 and
             Z is not at its upper limit
Then try the following actions in order:
 Action 1. Separable:
             up Z by 10%,
             down A by 2,
             set Material to "inco718",
             vary B, C, and D
 Action 2. Inseparable:
             set Z to 50,
             set Material to "tin125",
             up C by 100,
             vary C
 Action 3.  Sequential
             vary B
             vary C
             vary B, C, and D

This rule is weighted at: 90
```

where *X* is an output parameter that may be a part of the optimization function (e.g., turbine efficiency) or may have lower bounds (e.g., minimum flow angle), and *Y* and *Z* in the Conditions part may be input or output parameters.

This rule will be fired under the following circumstances: to increase *X* has become part of the current goal, the Conditions listed above are not violated, the weight of this rule is higher than other rules that can be fired, and all actions in this rule are not currently suspended because of previous repeated failures to increase *X* without violating other constraints. Engineous will try the actions in the order given.

The key word "separable" tells Engineous to vary one or more parameters within the action even if some others are not allowed to vary for this particular run. For Action 1, Engineous will increase *Z* by 10% of its current value if it has no upper bound, or 10% of the difference between its current value and its upper limit. It will subtract 2 from *A* and set *Material* to "inco718". Then it will pass *B*, *C*, and *D* as variable parameters to the optimization module in order to maximize *X*. If this action does not increase *X* without increasing the total amount of constraint violated, Engineous will restore the values of *Z*, *A*, and *Material* and go to Action 2. No part of Action 2, which is an "inseparable" action, will be taken if any one of its parameters is not allowed to vary or is suspended. "Sequential" actions are to be taken one at a time. If an action is successsful, Engineous will repeat it until it reaches the point of diminishing returns.

A key feature of Engineous is its capability of running CAE codes unmodified. CAE codes are executed as subprocesses rather than the common practice of treating them as subroutines because (1) source codes are seldom available for commercial codes, (2) in-house codes are modified frequently and need to be able to "plug in" easily, (3) many CAE codes will experience floating point exceptions during design exploration, and decoupling the CAE codes will allow Engineous to recover and continue without aborting the whole design process, and (4) stand-alone CAE codes can examine various trial designs in parallel (a utility was provided to distribute the analyses of trial configurations to a number of otherwise ideal workstations). Information is passed between Engineous and CAE codes through input/output files just as a human designer would pass it. An extensive utility was developed to allow users to show Engineous by example how to edit input and extract output values. Engineous can convert the examples to source codes without programming.

Design Optimization

The Engineous optimization module combines various optimization technologies in an "interdigitized" manner, so that they are tightly coupled and switcned from one to another at various stages of a single design run. (The term "interdigitized" is derived from the image of two clasped hands with fingers intertwined.) Individually, each approach has advantages and disadvantages. In combination, they complement each other in many ways.

Interdigitation of Expert Systems and Numerical Optimization

The design rule system shown in the previous section illustrates one way this interdigitation is used, i.e., to supplement incomplete knowledge by numerical optimization. A number of key parameters (*B*, *C*, and *D*) were identified as strongly related to the goal but how to change these parameters is not known (see Action 1). Therefore, they were passed to a numerical optimization package called ADS (Automated Design Synthesis [Vanderplaats, 1988]) when this rule was fired to determine the best set of values in conjunction with changes in other actions in this rule.

Another way to utilize this interdigitation is to use design knowledge to focus numerical optimization to first vary parameters most likely to have an impact on the objective function given certain conditions (see Action 3). It is often desirable to obtain the greatest gain in the least number of analysis runs, particularly if there are many parameters. Preventing the numerical optimization package from varying parameters known to have little impact for a given condition not only reduces the gradient calculation, but also reduces the approximation error in the construction of search directions. Additional rules with more parameters but lower weight may be added to search a larger parameter space when time permits.

The third common use of this concept is to prevent being trapped in local optima. If certain knowledge about the design space is available, the rule system can be used to jump around design spaces (e.g., Action 2 in the rule example, "up *C* by 100" plus "vary *C*", will repeatedly add 100 to the current *C* value and then ADS will vary *C* starting at the new value) or move in a manner to avoid being trapped in constraint boundaries.

Interdigitation of Numerical Optimization and Genetic Algorithms

For design problems with no design knowledge available and with either multiple optima or complex constraints, numerical optimization and a genetic algorithm may be combined to obtain a better global optimum with reasonable turnaround time. The interdigitation reduces the high computational cost by entering the genetic algorithm only after the more efficient exploration techniques of numerical optimization have been exhausted. Once a better design is found by the genetic algorithm, the interdigitized technique pursues this better design with numerical optimization techniques. When the numerical techniques again arrive at an impasse, the genetic algorithm is reseeded with the best designs discovered by numerical optimization and the past best designs from the previous genetic algorithm population. This seeding provides "fertile" schemata for propagation and combination within the genetic algorithm.

Customizable Interdigitation - Optimization Plan

In complex engineering systems, different optimization techniques are often needed in different problems, or different parts of a problem, as well as at different stages of design optimization (e.g., different techniques are used to meet constraints, obtain a local optimum, and explore global optima). To deal with this issue, an "optimization plan" concept was formulated by Powell [1990] to allow the interdigitation of a number of optimization techniques, switching from one to another within a single design. The current choices of techniques include expert systems, a collection of over 40 numerical optimization options, genetic algorithms, and simulated annealing, as well as a number of heuristic search techniques. A default plan which was found to be generally robust for a wide range of problems is provided. This plan makes use of design knowledge first and then gradually injects other more exploratory techniques. The best design from the last optimization technique, genetic algorithms, is used as a new starting design for the next run of the optimization plan (see Figure 1). Whenever the genetic algorithm is entered on subsequent runs of the optimization plan, the genetic algorithms population is reseeded. See Powell [1990] for a detailed description of this concept.

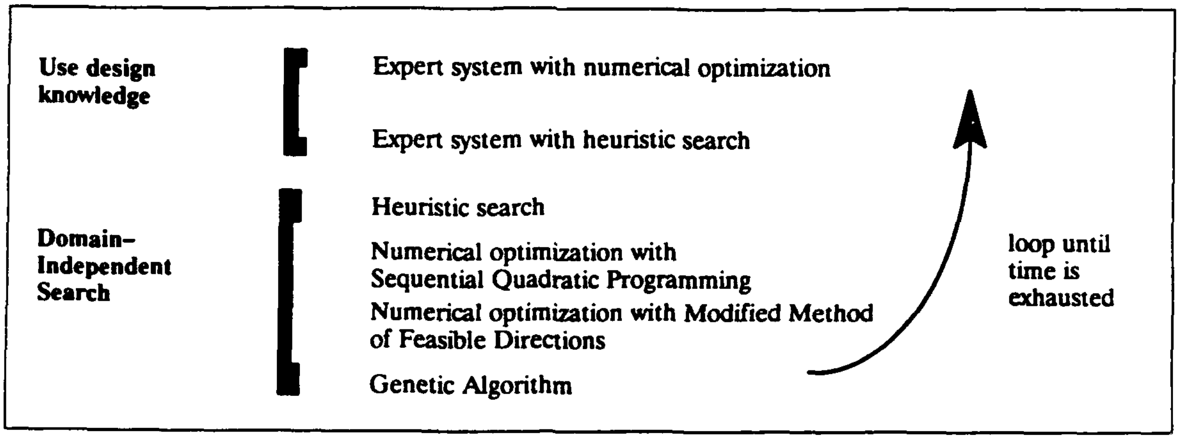

Use design knowledge
Expert system with numerical optimization
Expert system with heuristic search
Domain-Independent Search
Heuristic search
Numerical optimization with Sequential Quadratic Programming
Numerical optimization with Modified Method of Feasible Directions
Genetic Algorithm
loop until time is exhausted

Figure 1: Default optimization plan

Design Integration

In order to perform design integration, a number of capabilities are required, for example:

- Managing the relationship between parameters that are the interface between components or disciplines (e.g., the rpm of two components sharing the same shaft should always be the same)
- Allowing incremental building of a large application
- Facilitating the trade-off between components and between disciplines

These capabilities are provided in Engineous through the use of object-oriented programming and a simple constraint propagation mechanism. All entities in Engineous — including parameters, programs, design rules, search methods, etc. — are represented by objects. The inherent mechanism provides an efficient way to manage the complexity of a large-scale application. The simple constraint propagation is represented as methods activated by any change in parameters that are related to other parameters. A generic parameter-studies object is provided to study trade-offs. It can be used: (i) simply to vary some predefined parameters in a certain pattern, or (ii) to combine predefined patterns recursively to form a more complex study matrix, or (iii) at each complex matrix entry to redo the whole design optimization sequence with a new set of variable parameters. Some designers view this parameter studies feature as very useful.

The current Engineous system contains approximately 80,000 lines of Common Lisp, 10,000 lines of Fortran, and 25,000 lines of C code. A C++ based implementation replacing Lisp is being explored.

Historical Test Cases

A test set of six engineering optimization problems was used to validate that interdigitation is a more efficient and robust optimization technique than either genetic algorithms or numerical optimization used in isolation. The six engineering problems were part of an original test set of 30 problems selected by Sandgren [1977] to analyze the performance of 25 numerical optimization codes. The numerical optimization codes performed poorly on six test problems because of discontinuities, multimodality, gradient insensitivities, equality constraints, and scaling problems. Since no optimization code could solve more than two of these problems, Sandgren dropped the six engineering problems from his final comparative analysis. These six problems were selected for this analysis because they are a difficult test set, have known optima, and are representative of the types of parameter space problems that are found in real-world engineering problems (e.g., discontinuities, multimodality). The test problems are shown in Table 1.

The performance of Engineous using its interdigitation approach proved to be more robust and efficient than either numerical optimization using sequential quadratic programming, numerical optimization using the modified method of feasible directions, or a genetic algorithm used in isolation. (A detailed discussion of the performance of each technique on each problem is in Powell [1990].) The number of the problems solved within 5000 runs of the simulation code by the best numerical optimization technique, genetic algorithms and Engineous are shown in Table 2. A 5000-run limit was established to simulate the limited amount of time available for an engineer to obtain a design (e.g., a code that requires 10 seconds to run can be optimized in 13 hours). Overall, Engineous outperformed numerical optimization and genetic algorithms. For example, for a total relative error of .5, Engineous solved 5 problems compared to 1 for numerical optimization and 1 for genetic algorithms.

Problem	S	N	J	K	Feasible Starting Point	ASC	AOC	Characteristics
1. Chemical Reactor Design	#9	3	9	0	Yes	0	0	
2. Gear Ratio Selection	#13	5	4	0	Yes	0	0	Discontinuous objective function
3. Three-Stage Membrane Separation	#21	13	13	0	No	3	11	Small feasible region
4. Five-Stage Membrane Separation	#22	16	19	0	No	7	16	Small feasible region Poor scaling
5. Lathe	9	10	14	1	Yes	4	3	
6. Waste Water Treatment	#30	19	1	11	No	4	12	Large number of nonlinear equality constraints

S - Sandgren reference case J - Inequality constraints ASC-Active Side Constraints
N - Design variables K - Equality constraints AOC-Active Output Constraints

Table 1: Characteristics of engineering test set

Relative Error	Optimization Approach		
	E	NO	GA
.5	5	1	1
.25	4	1	1
.1	3	1	1
.075	2	1	1
.05	1	0	1
.01	0	0	1 (Sandgren #22)

Relative error $= \left| \frac{f(x) - f(x^*)}{f(x^*)} \right|$
where x* is optimal design

E - Engineous (no knowledge)
NO - Numerical Optimization
GA - Genetic Algorithm

Table 2: Number of historical test problems solved for each optimization approach for different relative errors

Engineous Applications

Back-to-back comparisons of designs using manual processes and designs using Engineous are available for a number of commercial product applications. The following examples are included here to indicate the levels of complexity Engineous is currently dealing with as well as the impact it has had on productivity.

Aircraft Engine Turbine Preliminary Design

A modern high-bypass engine turbine consists of multiple stages of stationary and rotating blade rows inside a cylindrical duct. A typical large transport engine turbine has 7 stages and over 700 parameters in the preliminary design phase; 100 of the input parameters are varied.

This application was implemented while Engineous was being developed. The time it takes to install this application using the current version of Engineous is estimated to be around one to two man-months.

A multi-stage low pressure (LP) turbine was selected as a good candidate for testing the applicability of Engineous in a real-world design environment with a real design project. For the particular turbine in question, a design optimization procedure had already been started in which a designer was optimizing the turbine with a goal of 0.75% efficiency improvement. The Engineous design was started in tandem with that work. The designer achieved a 0.5% improvement in design in 10 man-weeks while the Engineous design achieved a 0.92% improvement in one week.

The power of interdigitation is also shown with this turbine design application. Figure 2 shows the partial result of a thorough study for a new 2-stage turbine design. Over 30 ADS options with different search methods, gradient deltas, convergence criteria, and normalization were tested. The ADS work was stopped when no gain even as small as 10E-06 could be made. It can be seen that some ADS runs did much better than the expert system in this case. The performance of ADS was found to be highly dependent on the choice of ADS parameters, whereas an expert system always produces "good" results without tuning. Full interdigitation of all three search methods outperforms each of the three search methods by an efficiency gain of as much as 1%, a very significant number for turbine efficiency. The final turbine was an unconventional design with some of the parameter distributions opposite to what was done traditionally. Analysis of the optimization history shows that although the use of the genetic algorithm resulted in only a small gain in efficiency, it does appear to have pushed the optimization process away from being trapped in constraint boundaries so that the local hill climbing process could continue. Note that without the genetic algorithm the partial interdigitation of expert system and ADS produced a lower-performance (0.5% less than that of full interdigitation) but conventional turbine.

Molecular Electronic Structure Design

This design task was to locate the lowest energy state of a molecular electronic structure. The design has less than a dozen variable parameters. The analysis code is relatively slow and has to be executed on a remote mini-supercomputer.

It took one man-week of effort to solve a simplified case by hand iteration. It took half a day to couple the analysis code to Engineous, and one day to produce a similar solution. En-

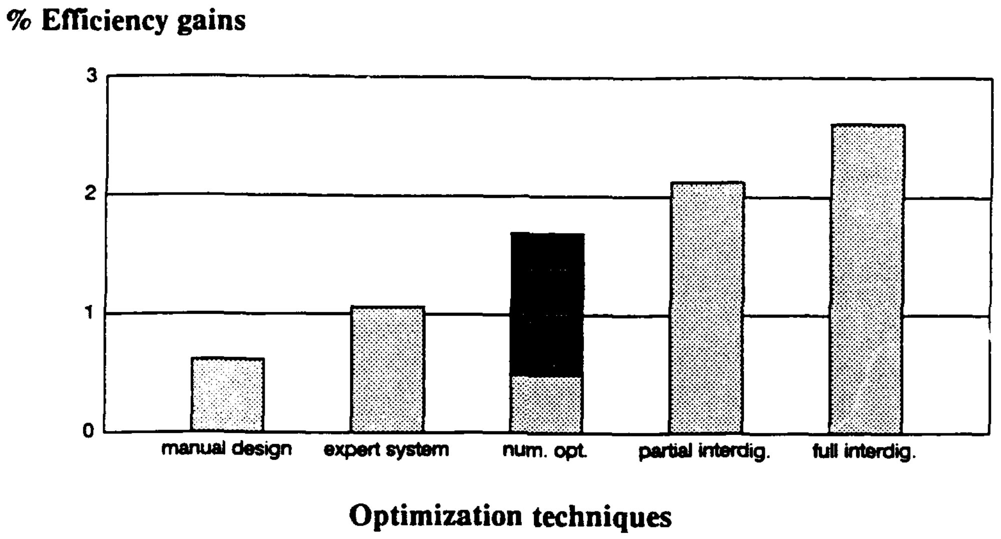


Figure 2: Efficiency gains vs. iteration for various search methods: expert system, numerical optimization (the heavily shaded region representing range of performance depends on ADS parameter settings), partial interdigitation (i.e., expert system and ADS), and full interdigitation (i.e., expert system, ADS, and genetic algorithm)

gineous has since been applied to a more complex case for this design task that could not have been solved by hand.

Cooling Fan Design

This is a very simple design task for Engineous, with only 44 parameters, 18 of which are input. The significance of this application is that the same problem has been solved with a pure OPS5 rule-based system, EXFAN [Tong, 1986]. The development of EXFAN from scratch took approximately 2 man-months. Reimplementing the same fan design with Engineous took half a day.

The design task is to determine the geometry, rotational speed, and air flow characteristics of a simple cooling fan. The objective is to minimize power consumption while maintaining a certain airflow. The quasi-3D fan analysis code uses 2D airfoil experimental data and requires aerodynamic knowledge to interpret the results. Ensuring a valid solution is the key issue.

The designs generated by Engineous are slightly better than those developed by design experts. The turnaround time is less than 10 minutes for Engineous and half a day for the human expert.

DC Motor Design

The design task is to determine a complete specification for an industrial large DC motor. This task has 180 parameters, over 70 of which are input. The first step is to identify the best configuration from a few hundred existing designs or generate a customized design if needed.

This is considered a simple problem for Engineous, which it took a few man-days to implement. The turnaround time of Engineous for this application is a few hours compared to a few man-weeks by the design expert.

Power Supply Design

Engineous has been coupled to HSPICE for testing its applicability to power supply design. A simple forward converter with a target voltage of 75 volts was designed both by a design expert and by Engineous. Engineous completed the design in 10 hours and obtained a 75.08 volts design; the design expert took approximately 3 weeks to obtain a 71.7 volts design. It took just a day to couple Engineous to this test problem.

Nuclear Fuel Lattice Design

The core of a nuclear reactor consists of a large number (hundreds) of fuel bundles. Each bundle may contain a square matrix of 50 to 75 fuel rods with different material compositions. Even though the current choice of rods is limited to a few dozen, the solution space is of the order of 10^{10} to 10^{15}. This is a demanding application because of the tight constraint of average uranium enrichment range over the bundle.

A simplified test problem was formulated to test the applicability of Engineous to this domain. This test problem is formulated as a uranium shuffling problem and numerical optimization was not applicable. The use of human design rules enabled Engineous to meet constraints in a few runs compared to a few hundred using a genetic algorithm. However, the genetic algorithm obtained significant improvement over the expert system in optimization. The resulting system obtains a preliminary fuel bundle design with a more optimized single design objective in a few days of elapsed computer time compared to approximately one week of manual effort for the same task.

Concurrent Aerodynamic and Mechanical Detailed Design of 3D Turbine Blades

This application is being used to guide the current development of Engineous. It illustrates the type of complex multidisciplinary design tasks Engineous was developed to solve. The process calls for completing a preliminary design of an aircraft engine turbine, then obtaining a complete detailed geometry of all the turbine blade rows that meets the aerodynamic requirement of the turbine as well as mechanical vibration and static stress constraints. This application involves approximately three dozen CAE codes, where some of the three-dimensional CAE codes, such as ANSYS and CAFD, are so complex that they may require a few man-weeks for a designer to analyze just one design scenario. A typical design cycle time, depending on the number of stages and complexity, is 12 to 24 man-months. Preliminary results show that a large part of this design task can be reduced to a few man weeks using Engineous. The CAE code flowchart is shown in Figure 3.

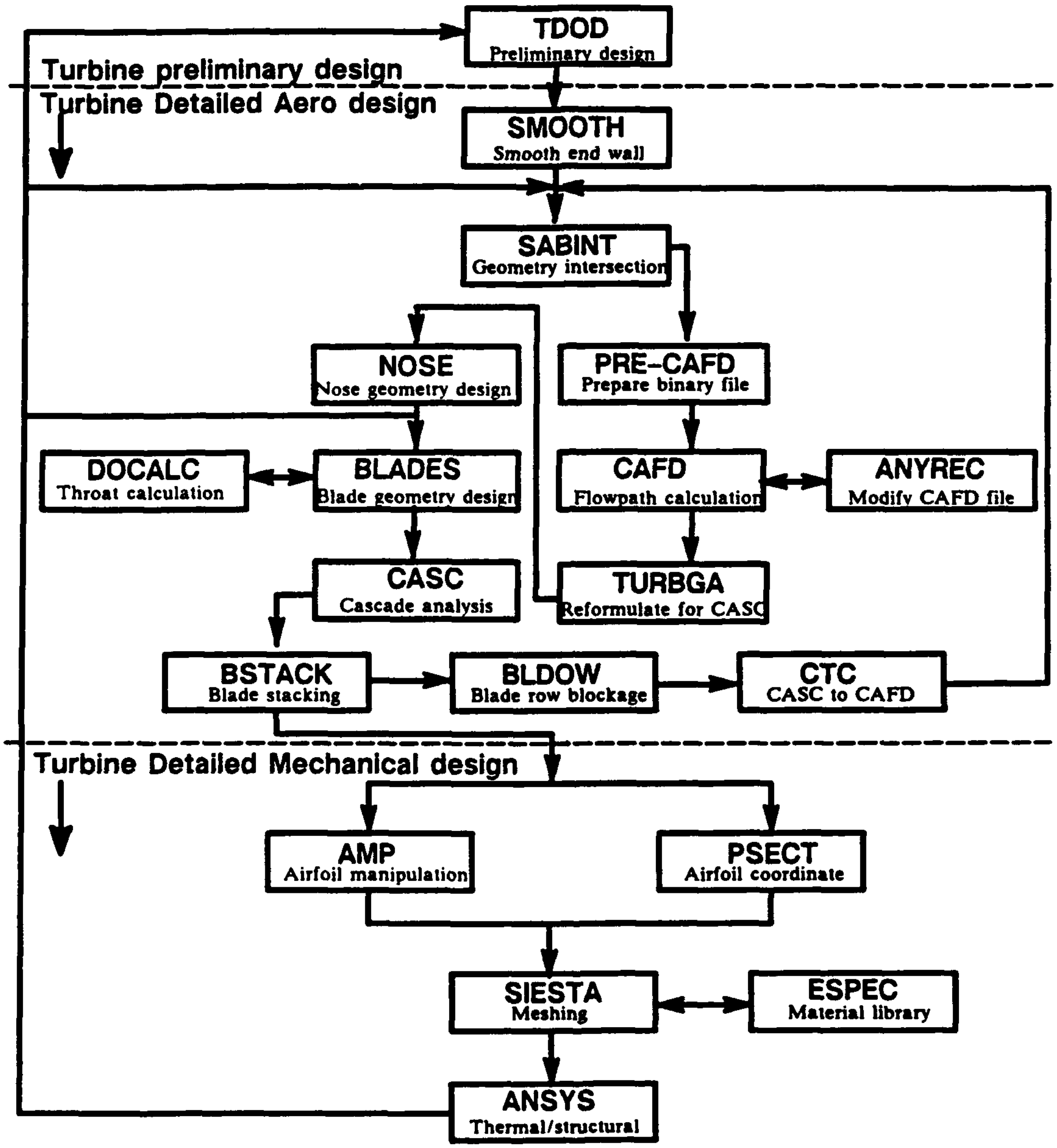


Figure 3: Turbine blade preliminary and detailed design with concurrent aerodynamic and mechanical analyses

Related Work

A number of other software systems employ some of the concepts described in this paper. For example, Kroo and Takai's PASS system [1988] provides an environment for aerodynamic design and trade-off studies with a limited choice of optimization techniques. This system is specifically built for one domain, and analyses are performed by subroutines hardwired to the environment. Bouchard, Kidwell, and Rogan's Engineer's Associate [1988] provides a generic framework to work with systems that can be represented by equations. Trade–off studies and a few generic optimizers are also provided. Both systems were developed for aerospace applications. The major difference between these systems and Engineous is that Engineous operates on an analysis code level rather than equation or subroutine levels. The need to drive various analysis codes without modification and the very large CPU requirement have added complexity to the Engineous environment. Also, because the application domain is more diverse, Engineous must provide much more optimization capability.

Conclusion

The current version of Engineous has demonstrated the profound impact such a system can have on productivity and performance. The interdigitation of numerical and symbolic techniques allows Engineous to solve a wide range of problems, from well–understood tasks where there is ample knowledge to new design problems; from smooth to rough objective functions; from aiming for conservative design to exploring new concepts; and from problems with real parameters only to complex problems with a mixture of real, integer, and symbolic parameters.

With the rapid increase of computational power and the advance of numerical simulation methods, the number and complexity of engineering products that can be analyzed accurately will be increased rapidly and analysis turnaround time reduced substantially. The need for human intervention in computation–based designs is currently a productivity bottleneck, preventing many powerful numerical simulation modules from being used to their full potential. Application of a system like Engineous can not only make the design process more productive but also, by relieving designers of the need to carry out tedious, iterative procedures, make it more fun.

Acknowledgments

A large part of the current Engineous development is headed by Dr. David Powell of GE Corporate Research and Development Center, with contributions from Dr. Mark Kolb, Messrs. Hauhua Lee, Sanjay Goel, Randy Harms, Scott Huston, Gill Derge, and Steve Jacobson. Mr. Brent Gregory, manager of Turbine Aero and Cooling Technologies at GE Aircraft Engine, contributed greatly to the success of the turbine preliminary design project. Professor Garret Vanderplaats provided his ADS expertise. Engineous technology development is funded by the GE Research and Development Center. The turbine blade concurrent design problem is funded by DARPA.

References

Adeli, H. (1986). "Artificial intelligence in structural engineering." *Engineering Analysis 3.*

Ban den Bout, D.E., and Miller, T.K. *Graph Partitioning Using Annealed Neural Networks.* North Carolina State University Center for Communications and Signal Processing and Computer Systems Laboratory, Raleigh, NC.

Bouchard, E.E., Kidwell, G.H., and Rogan, J.E. (1988). "The application of artificial intelligence technology to aeronautical system design." AIAA-88-4426, AIAA, AHS, and ASEE, Aircraft Design, Systems and Operations Meeting, Atlanta, Georgia, September 7-9.

Gabriele, G.A. (1988). "The generalized reduced-gradient method for engineering optimization." In R. Levary (Ed.), *Engineering Design.* Pp. 126-144. New York: Elsevier Science Publishing Co.

Gero, J. (1988). "Development of a knowledge-based system for structural optimization." *Structural Optimization.* The Hague: Kluwer Academic Publishers.

Giles, M., Drela, M., and Thompkins, W. (1985). "Newton solution of direct and inverse transonic Euler equations," Proceedings of the AIAA 7th Computational Fluid Dynamics Conference, Cincinnati, Ohio, July 15-17, 390-402.

Goldberg, D.E. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning.* Reading, MA: Addison-Wesley.

Holland,, J.H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: Univ. of Michigan Press.

Jabri, M. (1987). "Implementation of a knowledge base for interpreting and driving integrated circuit floor planning algorithms." *AI in Engineering* 2.

Kirkpatrick, C., Gelatt, C., and Vecchi, M. (1981). "Optimization by simulated annealing." *Science,* 220 (4598) 671-680.

Kroo, I., and Takai, M. (1988). "A quasi-procedural knowledge-based system for aircraft design." AIAA-88-4428, AIAA, AHS, and ASEE, Aircraft Design, Systems and Operations Meeting, Atlanta, Georgia, September 7-9.

Powell,D.J. (1990). Inter-GEN: A hybrid approach to engineering optimization. Rensselaer Polytechnic Institute Ph.D. thesis, Troy, NY.

Sandgren, E. (1977). The utility of nonlinear programming algorithms. Purdue University Ph.D. thesis, West Lafayette, IN.

Tong, S.S. (1984) Procedures for accurate inviscid flow simulations and profile refinement of turbomachinery cascades. Massachusetts Institute of Technology, Department of Aeronautics and Astronautics, Ph.D thesis, Cambridge, MA.

Tong,S.S. (1986). "Coupling artificial intelligence and numerical computation for engineering design." AIAA-86-0242, AIAA 24th Aerospace Sciences Meeting, Reno, Nevada, January 6-9.

Vanderplaats, G.N. (1984). *Numerical Optimization Techniques for Engineering Design: With Applications.* New York: McGraw-Hill.

Vanderplaats, G.N. (1988). *ADS — A Fortran Program for Automated Design Synthesis.* Santa Barbara, CA: Engineering Design Optimization, Inc.

Winston, P.H. (1984). *Artificial Intelligence.* 2nd Ed. Reading, MA: Addison-Wesley. Pp. 88-101.

Zhou, H.H. (1987). CSM: A genetic classifier system with memory for learning by analogy. Vanderbilt University Ph.D. thesis, Nashville, TN.

Symbiotic Computation: Opportunities and Complications

by

K.S. Booth[a] and W. Morven Gentleman[b]

[a]University of British Columbia
Vancouver, B.C.
Canada

[b]National Research Council of Canada
Inst. for Information Technology
Ottawa, Ontario
Canada

Extended Abstract

Personal computers and personal workstations have become the normal way scientists and engineers do computing. However, not all computations can be done on the local machine. Programs that are too large, programs that depend on unique software or hardware resources, programs that are proprietary, programs that are shared subject to licencing restrictions on usage, datasets that are too large, and datasets that exist in unique locations are all reasons for having to run a program on a remote computer. The local computer still is used, but it is used only to provide access to the remote computer through a network or across communication lines.

At issue is what "provide access" means. At one time, it meant that the local personal computer or personal workstation emulated a character mode terminal such as a VT100 or a graphics terminal such as a Tektronix 4010. The modern version of this is the X terminal, which provides locally a multi-window environment that the remote program (usually running on a mainframe, but sometimes on another personal computer or workstation) can control. In such a terminal paradigm, the terminal "logs onto" a remote host computer, and the preparation of the run of the program, the interaction with the running program, and the analysis of the program results are all done on the remote host computer, using the host's computational resources. The remote program controls what is displayed locally and how local input is accepted, that is, the user interface is determined by the remote program not the local environment on the workstation. The remote program builds this interface out of local input/output services that are available independent of which remote program is used, and the communications protocol is a standard that is used by all applications that use the remote computer.

The symbiotic computation paradigm is quite different. In it, the computation is partitioned between a program running on the local computer and a program running on the remote computer. The user interface is completely determined by the program on the local computer, and the protocol for communications between the local and remote computers is specific to that application. In particular, what the user at the keyboard types is not necessarily what is sent to the remote program, and what the remote program sends is not necessarily what the user sees on the screen. Commercial terminal emulators on personal computers almost all provide some form of scripting language for communication control, which is the simplest instance of symbiotic computation. Symbiotic computation has developed largely for two reasons: first, because with almost unbounded processing capability now available at both the local and remote computers network bandwidth can be minimized using the symbiotic approach, but second, and more importantly, because the binding times of the two programs can be different the user can customize the user interface to his own needs, even when the remote program is not under his control (i.e. there is no need for every user of the remote program to see the same interface). The trend in the data processing world to client/server systems provide other examples of symbiotic computing.

This paper explores how such a symbiotic computation paradigm can be used, what functionality it must provide, and what programming problems arise in trying to make the local and remote programs collaborate effectively. To look at how symbiotic computation can be used, we need to keep in mind the objectives of why it is used. Historically, the most important has been to hide unnecessary complexity and inflexibility imposed by the remote program, by the need to deal with several remote programs either sequentially or concurrently, and even by the need to deal with several remote systems. The second most important objective has been data compression to facilitate remote access over limited bandwidth lines. Display of computed data in ways intuitive and insightful to a particular user on a particular run has also been important. The theme appears both for input and output that the user has more than one focus of concern and that these are largely unrelated, so that multiple contexts and even multiple threads of execution are required to overcome the unnecessary sequentiality imposed by glass teletype terminals. Multiple windows are a key advance. Asynchronous, or at least out of expected sequence, actions such as probes initiated by the user or exceptions reported by the remote program especially benefit from this treatment.

The data input required to specify the particular problem to be solved is a major task for many scientific and engineering computations, and neither offline nor conventional online data entry is completely satisfactory. Symbiotic computation can be used to address this in several ways. Providing interactive help to explain input requirements, validating input data against constraints, allowing a spreadsheet-like ability to enter expressions rather than just numeric values, making links to databases and to other programs, facilitating annotations of why some value was used, and even encouraging non-keyboard options for data entry from menus or analog displays are all techniques to reduce the probability of accidentally setting up and running the wrong problem. Logging the input actually used is important for identifying the point in problem space to which the results correspond, but it can also be useful if the course of execution turns out to require unanticipated entry of further data, because this often necessitates terminating the run and reinitiating it once the required data has been determined, and of course it is desirable not to have to re-enter data once given.

For many scientific and engineering programs, the complete computed results are too massive to move to the local computer, nevertheless, key monitoring data is commonly displayed on a terminal, and symbiotic computation facilitates recording such data, doing analysis and presenting the data in various ways such as summaries, history, replay, or visualization. Data such as this can also drive expert system suggestions for further analysis or other problems to try.

The programming issues that arise in all this are partly how to make life easier both when programming the remote computational program and when programming the local interface, as well as how to cope with the independent evolution of these programs. The programming style to cope with asynchronous communication between the programs is particularly significant. Object oriented technology, direct manipulation, and scripting play an important part. Communications protocol should not be hidden, but should be part of the specification of the interface to the remote computation. In many cases this will mean that the remote computation must be structured as a server, capable of responding asynchronously to requests for input, output, and status information. The local computation will also take on this same organization since it will have to respond to similar requests from both the remote computation and the local user.

Problem Solving Using *Mathematica*

Paul C. Abbott
Wolfram Research (UK) Ltd.
P. O. Box 114
Abingdon
OXON OX13 6TG
United Kingdom

Abstract

Mathematica is a general system for numerical, symbolic and graphical computation which can be used both as an interactive calculation tool and as a programming language. Its capabilities include arbitrary precision arithmetic, matrix manipulation symbolic algebra, integration, differentiation, power series, two-dimensional plots, contour plots and shaded colour three-dimensional pictures.

Through its integration of all the elements necessary for quantitative analysis, its wide availability, portability, large user-base and consistent system design, *Mathematica* provides the most complete problem solving environment presently available.

Mathematica runs on a wide range of machines including the Macintosh, PC, NeXT, Sun, Sony, Dec, HP/Apollo, IBM, and Convex computers and *Mathematica* code on all these machines is completely compatible. Of course, the smaller machines are limited in their capabilities by speed and memory but the *Mathematica* system design permits computational scaling. Furthermore, the separation of the front end and computational kernel allows small machines to interface to powerful computational engines in a transparent fashion.

This paper provides a brief overview of the capabilities of *Mathematica* through a few diverse examples ranging from solving the Kepler equation to visualizing random walks. A complete description of *Mathematica* is found in [Wolfram 91].

Introduction

Mathematica has a large range of built-in functions necessary for high-level problem solving including arbitrary precision arithmetic, *e.g.*,

```
85!

2817104114380550276949479442260611594800566343305742064051019127525 6\

  0026159795933451040286452340924018275123200000000000000000000
```

special functions in the complex plane, *e.g.*,

```
N[BesselJ[1, 3 + 2 I], 50]

0.78014884857925378451798593167023711206599266200017 -

  1.2609820602388484315992864301761607095709315523677 I
```

a complete symbolic language including differentiation and integration, *e.g.*,

```
Integrate[ExpIntegralEi[-a x] Sin[b x], {x,0,Infinity}]

                        2    2
a (2 Log[a] - Log[a  + b ])
-------------------------------
        2 b Abs[a]
```

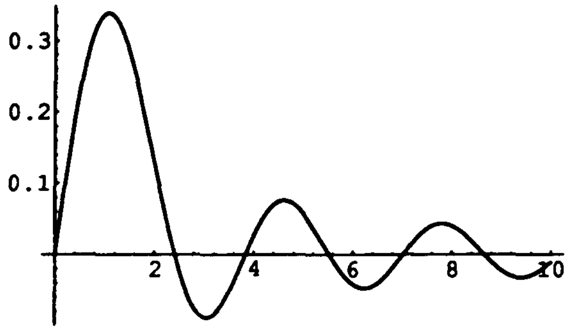
trigonometric identities, *e.g.*,

```
Factor[3 Cos[x]/128 - Cos[3x]/64 - Cos[5x]/64 + Cos[7x]/256 +
    Cos[9x]/256, Trig -> True]

      5      4
Cos[x]  Sin[x]
```

and visualization, *e.g.*,

```
Plot[BesselJ[0,x] BesselJ[1,x], {x,0,10}];
```



In the following sections, examples from a range of areas are considered that attempt to give a feeling for the application and overall capabilities of *Mathematica*.

Kepler Equation

The Kepler equation arises in celestial mechanics. The equation `s = u + e Sin[s]` where `E` is a function of both `u` and `e`, with `e` regarded as a small quantity, is easily solved using power series. Note that the *Mathematica* syntax naturally lends itself to the solution of this problem.

Series Solution

Introducing the quantity `a[k] = e Sin[u + a[k-1]] + O[e]^(k+1)` then, in the limit `k -> Infinity`, `a = e Sin[u + a]`. Hence `s = u + a` as then `a + u = u + e Sin[u + a]` and finally `s = u + e Sin[s]`.

Defining the initial condition,

```
a[0] := 0
```

and the iteration,

```
a[k_] := a[k] = e Sin[u + a[k-1]] + O[e]^(k+1)
```

where *dynamic programming* is used to store all the intermediate computations, we obtain, *e.g.*,

```
a[3]
```

```
                                                                     3
                              2          2           Sin[u]      3       4
Sin[u] e + Cos[u] Sin[u] e  + (Cos[u]  Sin[u] - ------) e  + O[e]
                                                         2
```

To simplify the expressions we require a Fourier-type expansion:

```
expand[exp_] := Expand[Normal[exp], Trig->True]
```

Here is the sixth term in the expansion:

```
a[6] // expand
```

```
                3               5               2                4
               e  Sin[u]    e  Sin[u]    e  Sin[2 u]    e  Sin[2 u]
e Sin[u] - --------- + --------- + ----------- - ----------- +
                8               192             2                6

  6                 3                 5                  4
 e  Sin[2 u]    3 e  Sin[3 u]    27 e  Sin[3 u]    e  Sin[4 u]
 ----------- + ------------- - -------------- + ----------- -
     48                8                128                3

    6                  5                 6
 4 e  Sin[4 u]    125 e  Sin[5 u]    27 e  Sin[6 u]
 ------------- + --------------- + --------------
      15                 384                80
```

It is straightforward to collect together all the terms in **Sin[n u]**:

```
Collect[%, Table[Sin[n u], {n,1,6}]]
```

```
      3    5                  2    4    6
     e    e                   e    e    e
(e - -- + ---) Sin[u] + (-- - -- + --) Sin[2 u] +
     8    192                  2    6    48

   3        5                4      6
 3 e    27 e                e    4 e
(---- - -----) Sin[3 u] + (-- - ----) Sin[4 u] +
  8      128                 3    15

       5            6
 125 e  Sin[5 u]   27 e  Sin[6 u]
 --------------- + --------------
      384                80
```

Formal Solution

The standard formal solution involving Bessel functions is easily verified up to any desired order, *e.g.*,

```
s = u + 2 Sum[BesselJ[n, n e] Sin[n u]/n, {n,1,4}] + O[e]^4 // Simplify
```

```
                           2                    3
                 Sin[2 u] e    (-Sin[u] + 3 Sin[3 u]) e        4
u + Sin[u] e + ----------- + ------------------------- + O[e]
                    2                     8
```

which is to be compared to `u + e Sin[s]`:

```
u + e Sin[s] // Simplify
```

```
                           2                    3
                 Sin[2 u] e    (-Sin[u] + 3 Sin[3 u]) e
u + Sin[u] e + ----------- + ------------------------- +
                    2                     8

  -Sin[2 u]   Sin[4 u]   4      5
 (--------- + --------) e  + O[e]
      6          3
```

It is important to note that not only are the numerical properties of the whole range of the special functions of mathematical physics built-in but that *Mathematica* can also compute other properties such as series expansions and differential recurrence relations, *e.g.*,

```
D[BesselJ[n, x], x]
```

```
BesselJ[-1 + n, x] - BesselJ[1 + n, x]
--------------------------------------
                  2
```

Random Walks

The concept of the random walk is fundamental to statistical physics. One interesting application of random walks is to the Black-Scholes model used in stock trading.

The following examples illustrate random walks in one, two and three dimensions. List operations and plotting routines can be combined to yield concise programs for generating random walks. The *Mathematica* language is both high-level and natural and the extension to higher dimensions follows straightforwardly.

Cumulative sums can be computed using the `FoldList` command, *e.g.*,

```
FoldList[Plus, 0, {a,b,c,d,e}]
```

```
{0, a, a + b, a + b + c, a + b + c + d, a + b + c + d + e}
```

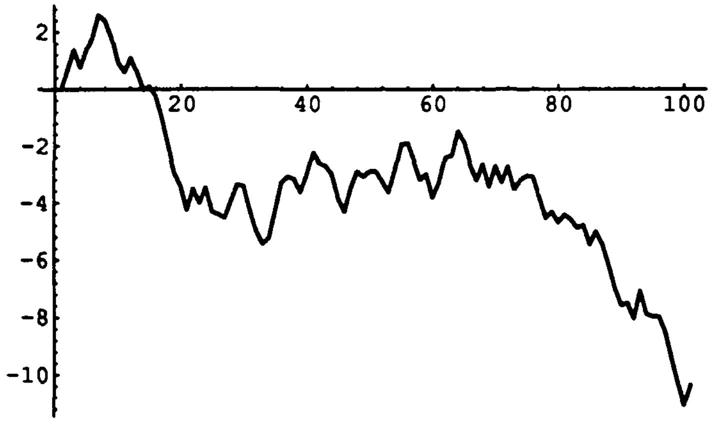
One-Dimensional Random Walk

Starting at the origin (`0`), `100` random real numbers in the range `{-1, 1}` are selected and cumulatively summed:

```
FoldList[Plus, 0, Table[Random[Real, {-1, 1}], {100}]];
```

The (suppressed) plot of the distance from the origin as the number of steps increase is given by

```
oned = ListPlot[%, PlotJoined -> True];
```



Two-Dimensional Random Walk

The extension to the two-dimensional case is straightforward:

```
FoldList[Plus, {0, 0}, Table[Random[Real, {-1, 1}], {100}, {2}]];
```

Here the graphical output is suppressed:

```
twod = ListPlot[%, PlotJoined -> True, AspectRatio -> 1,
                   DisplayFunction -> Identity];
```

Three-Dimensional Random Walk

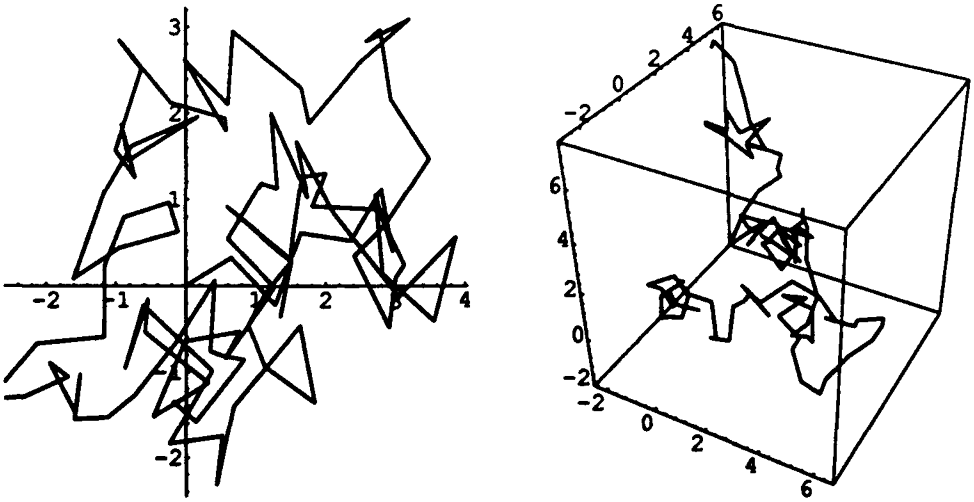
Finally in three dimensions:

```
FoldList[Plus, {0, 0, 0}, Table[Random[Real, {-1, 1}], {100}, {3}]];

threed = Show[Graphics3D[Line[%]], Axes -> True, DisplayFunction -> Identity];
```

and now we show an array of the two- and three-dimensional graphics:

```
Show[GraphicsArray[{twod, threed}], DisplayFunction -> $DisplayFunction];
```



Polynomials and Matrices

In this section, two methods for producing random polynomials show dramatically varying behaviour. *Mathematica*'s graphics are used to enhance intuition and understanding.

Random Polynomials

Consider the following definition of a polynomial with uniformly distributed random coefficients:

```
RandomPolynomial[n_, x_] := Table[x^i, {i,0,n}].Table[Random[], {n+1}]
```

e.g.,

```
RandomPolynomial[3,x]
```

```
                                  2             3
0.167616 + 0.205668 x + 0.400681 x  + 0.610133 x
```

The roots of a random polynomial of, *e.g.*, size 50 are given by

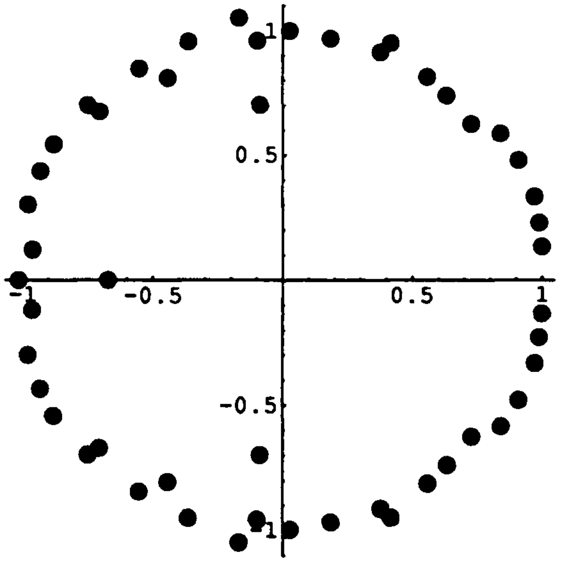
```
roots = x /. NSolve[RandomPolynomial[50, x] == 0, x];
```

and the real and imaginary components can be projected out using

```
argand = {Re[#], Im[#]}& /@ roots;
```

which permits visualization:

```
ListPlot[argand, PlotStyle -> PointSize[0.03], AspectRatio -> 1];
```



It is apparent that the eigenvalues lie approximately on the unit circle in the complex plane.

Random Matrices

Consider the following definition of square matrix with random real entries uniformly distributed between 0 and 1:

```
random[n_] := Table[Random[], {n},{n}]
```

For a `100 x 100` matrix,

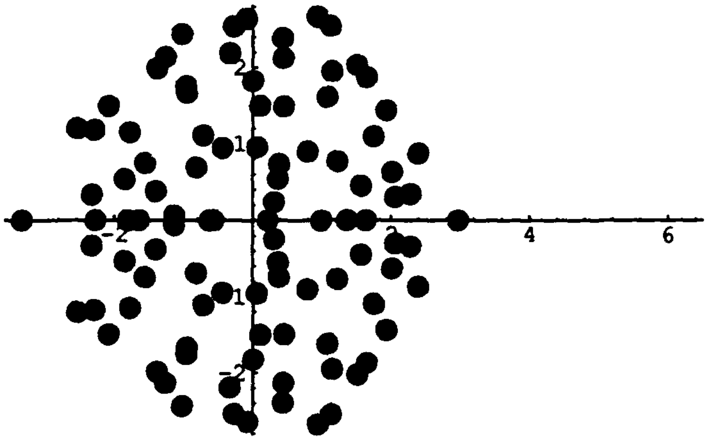
```
matrix = random[100];
```

its eigenvalues,

```
evalues = Eigenvalues[matrix];
```

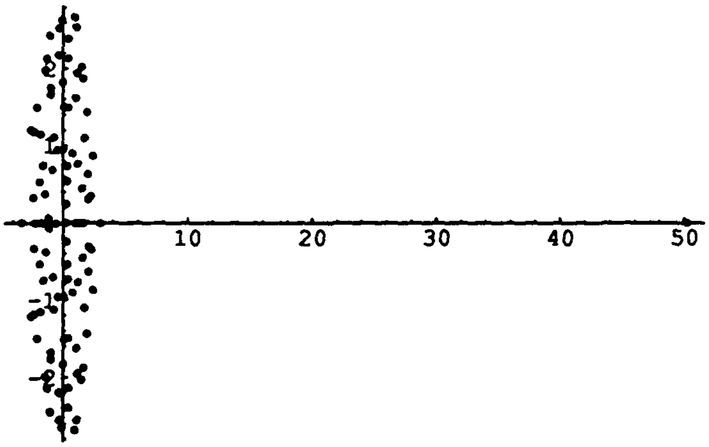
can be visualized using

```
argand = {Re[#], Im[#]}& /@ evalues;
ListPlot[argand, PlotStyle -> PointSize[0.03]];
```



We see that the large majority of the eigenvalues fit inside a circle centred on the origin. However, setting `PlotRange -> All` reveals that one of the eigenvalues is at the value of approximately `n/2` where `n` is the size of the matrix:

```
ListPlot[argand, PlotStyle -> PointSize[0.01], PlotRange -> All];
```



This example shows that *Mathematica* handles random numbers, matrices and complex numbers in a unified fashion and this makes it a good investigative tool aiding our physical and mathematical intuition. Note that the probability that two very large integer matrices have relatively prime determinants has recently been computed using *Mathematica* [Vardi 91] and the result is, like the situation in the above example, quite different to the probability that two large integers are relatively prime.

Electric Circuit

Since *Mathematica* has the capability to solve systems of numerical differential equations, many physical problems can be examined. For example, consider a pair of non-linear differential equations describing an electric circuit [Crandall 91, Levy 91],

```
equation1 = v[t] == r i[t] + k i[t]^b + l i'[t];
```

and

```
equation2 = v'[t] == -i[t]/c;
```

Supplying the circuit parameters; **c** (capacitance) , **r** (resistance), **l** (inductance), and the model parameters **k** and **b**,

```
parameters = {c->20 10^-6, r -> 2, l -> 10^-4, k -> 70, b -> 0.38};
```

to the equations,

```
equations = {equation1, equation2} /. parameters

                     0.38              i'[t]
{v[t] == 70 i[t]          + 2 i[t] + -----, v'[t] == -50000 i[t]}
                                     10000
```

specifying the initial conditions,

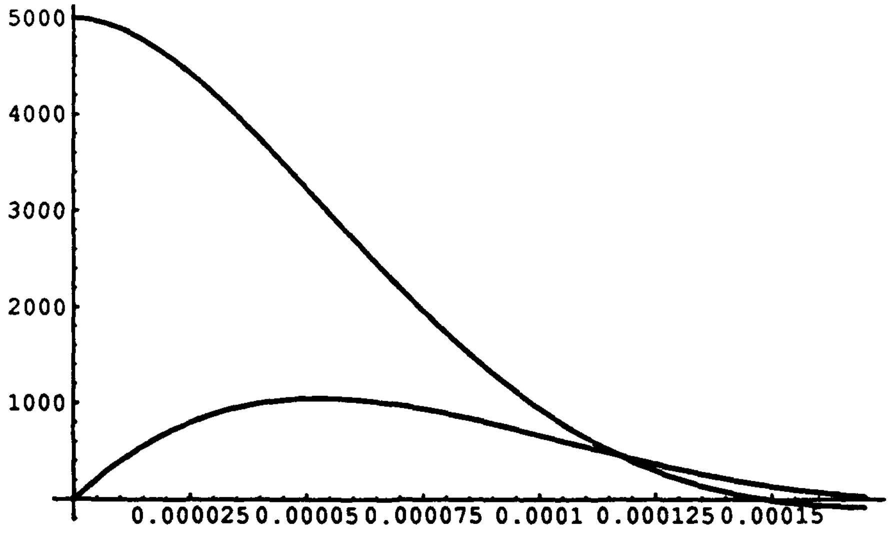
```
initial = {v[0] == 5000, i[0] == 0};
```

it is straightforward to obtain the solution for the first 170 μs,

```
solution = NDSolve[Join[equations, initial], {v[t], i[t]}, {t,0,0.00017}];
```

Here is a plot of the voltage and current as a function of time:

```
plot = Plot[Evaluate[{v[t], i[t]} /. solution], {t,0,0.00017}];
```



It is apparent that the modeling of quite complicated physical systems is quite straightforward since the numerical and graphical packages are built in.

Recurrence Relations

Suppose that one has a recurrence relation, *e.g.*,

```
n p[n]  == (n-1) p[n-1] + p[n-2]

n p[n] == p[-2 + n] + (-1 + n) p[-1 + n]
```

with the boundary conditions

```
p[0] = 1; p[1] = 0;
```

The general solution to a very large class of recurrence equations is possible using the **RSolve** package, implemented in the *Mathematica* programming language [Maeder 91], and loaded using the command

```
<<DiscreteMath`RSolve`
```

Entering the recurrence and the boundary conditions, one obtains the generating function:

```
gf = GeneratingFunction[{(n+2) p[n+2] - (n+1) p[n+1] - p[n] == 0,
        p[0] == 1, p[1] == 0}, p[n], n, z]

            1
{{------------}}
    z
  E   (-1 + z)
```

The series expansion about **z = 0** of this expression is

```
gf + O[z]^7

        2    3     4      5       6
       z    z   3 z   11 z   53 z         7
{{1 + -- + -- + ---- + ----- + ----- + O[z] }}
       2    3    8      30     144
```

A much more difficult question is the *general* term of this expansion. This is found using the **SeriesTerm** operator:

```
gt = SeriesTerm[gf, {z,0,n}]

       K[1]
   (-1)      If[n - K[1] >= 0, 1, 0]
{{----------------------------------}}
                 K[1]!
```

where **K[1]** denotes the (first) summation index. Rewriting the summand as

```
summand[n_, k_] = (% /. K[1] -> k);
```

permits the evaluation of a particular term using

```
term[n_] := Sum[summand[n, k], {k,0,n}]
```

For example,

```
term[5]

  11
{{——}}
  30
```

which agrees with the series expansion and the recurrence relation. Quite a broad class of recurrence equations can be solved using `RSolve`.

Conclusion

The integration of numerical, symbolic and graphical capabilities with a consistent syntax and natural programming language assists the modelling of physical problems. Although *Mathematica* is an interpreted language, which implies that it will be slower than a compiled language, it has several advantages: It is an easy language to learn because it is "natural" due to its high-level functionality; Numerical functions in *Mathematica* now "compile" (function compilation) and so the speed penalty is reduced; It is powerful because of its extensiblility and is an ideal platform for prototyping ideas – often a detailed analysis of a problem will indicate a better method of solution and this can immediately be implemented in *Mathematica* through its high-level programming language.

Through the use of symbolic manipulation, arbitrary precision arithmetic and graphics, the effects of computational errors and uncertainties in data can easily be ascertained. The language is application-oriented as it includes such a variety of control structures, functions and operations that many problems can be coded essentially "as is". The use of pattern-matching, coupled with rule based capabilities, provides a very powerful and easy to use programming environment.

References

Crandall, R., *Mathematica for the Sciences*, Addison-Wesley, 1991

Levy, S. (ed.), The *Mathematica Journal*, Addison-Wesley, 1991

Maeder, R., *Programming in Mathematica,* Addison-Wesley, 1991

Vardi, I., *Computational Recreations in Mathematica,* Addison-Wesley, 1991

Wolfram, S., *Mathematica: A System for Doing Mathematics by Computer*,
Addison-Wesley, 1991

Accuracy Control and Estimation, Self-Validating Systems and Software Environments for Scientific computation

S. M. Rump, Hamburg

Traditional numerical algorithms do, in general, produce very good approximations to the true solution of the posed problem. There are very efficient techniques for backward error analysis and also the methods of traditional numerical analysis for forward error analysis are on a very high level (condition estimators etc.).

The methods for estimating the true error of the computed solution (w.r.t. the true solution of the given problem) are, in general, estimators. They proved to be stable, pitfalls are rare. If it would be possible, however, to construct algorithms being comparable in speed with traditional numerical algorithms and giving validated bounds for the true solution of the given problem, this would be interesting, at least for critical problems.

Before we say that such algorithms exist for a number of standard numerical problems we must say some words on the "solution of a given problem". The point is that in the computer we frequently do not have *the original, practical problem* we want to solve but first a discretisation of it and second a rounded version of it (omitting tolerances inherent to the data of the problem). For example Bellman [Bel75] writes "Considering the many assumptions that go into the construction of mathematical models, the many uncertainties that are always present, we must view with some suspicion at any particular prediction. One way to contain confidence is to test the consequences of various changes in the basic parameters." Hence a "validated solution" must not only comprise of guaranteed error bounds for the solution of the particular problem stored in memory but also of (validated) information on the sensitivity of this solution w.r.t. perturbations in the input data.

Usually such a sensitivity analysis yields bounds for the amplification factors depicting the dependency of a particular component of the solution on variations of a particular input parameter. This bears e.g. the advantage that system zeros remain unchanged (see also [ArDeDu89]). It may happen, however, that discontinuous changes of the solution occur like in LP-problems in the basisinstable case. In this case the output comprises of a list of solutions being optimal with respect to some specific problem with the input tolerances (see [Ja85], [JaRu90]).

It is obvious that the computation of validated bounds requires a precisely defined computed arithmetic, preferably with smallest errors possible for the individual operations. A major step towards this goal is Kulisch's definition of the computer arithmetic and the IEEE 754 binary floating-point standard, the latter nowadays being implemented in many computers [IEEE86]. The standard also comprises of directed rounding modes. It suffices to produce validated answers; better and sharper results are obtained, however, using the arithmetic proposed by Kulisch [Ku76,81].

Algorithms for computing validated bounds with the described properties including a complete sensitivity information have been developed for a number of standard numerical problems such as general linear systems, eigenproblems, polynomial zeros, linear programming problems including the basisinstable case, general systems of nonlinear systems and others [Ru80,83,90], [ACR86], [Ar86].

Especially the automatical and validated sensitivity analysis offers great advantages to the user. The sensitivity allows to check the problem rather than the solution and a backward engineering: What accuracy is necessary for the data to meet specified tolerances in the components of the solution?

A (very) extreme example is the following system of linear equations:

$$\begin{aligned}
-367296\cdot t - 43199\cdot u + 519436\cdot v - 954302\cdot w &= 1\\
259718\cdot t - 477151\cdot u - 367295\cdot v - 1043199\cdot w &= 1\\
886731\cdot t + 88897\cdot u - 1254026\cdot v - 1132096\cdot w &= 1\\
627013\cdot t + 566048\cdot u - 886732\cdot v + 911103\cdot w &= 0.
\end{aligned}$$

The correct solution is

$$\begin{aligned}
t &= 8.86731088897 \cdot 10^{17}\\
u &= 8.86731088897 \cdot 10^{11}\\
v &= 6.27013566048 \cdot 10^{17}\\
w &= 6.27013566048 \cdot 10^{11}.
\end{aligned}$$

However, a relative perturbation of any matrix component by 10^{-34} alters the first figure of the solution. The sensitivity of the linear system is 10^{34}.

We will especially present some new ideas for a fast linear system solver producing validated bounds. This is in fact a hybrid algorithm. The fastest version needs only $n^3/3$ floating-point operations plus $0(n^2)$ interval operations. It is therefore as fast as traditional Gaussian elimination and works for moderate condition numbers. If the algorithm fails it gives an internal error message and switches automatically to the next algorithm in the series requiring additional $n^3/6$ floating-point operations. It never happens under no circumstances that a false answer is produced. The failing of any of our algorithms with result verification is monitored by an appropriate error message saying that the precision in use is not sufficient for the method to solve the problem.

The hybrid algorithm for linear systems comprises of 5 steps each requiring $n^3/3$, $n^3/2$, $2n^3/3$, n^3 and $2n^3$ floating-point operations in total, resp. plus $0(n^2)$ interval operations where the *additional* effort going from one stage to the next is exactly the difference of the given two numbers. In others words each algorithm can fully make use of what has been achieved by its predecessor.

Furthermore algorithms will be presented for solving large band and sparse linear systems with symmetric positive definite and with M-matrix. The presented algorithms require as many floating-point operations as a traditional LU-decomposition in the nonsymmetric or an LDL^T- or Cholesly-decomposition in the symmetric case plus some $2n^2$ interval operations.

In order to achieve validated results on a computer an adequate programming language is necessary as well.

Interesting enough a well-spread programming like Pascal requires few additional concepts to gain significantly for numerical and validated programming. Moreover, from a compiler point of view those concepts are not difficult to implement.

The concept of units, not included in Standard Pascal, is already part of many modern implementations. Name overloading which means that two functions and/or procedures may have the same name if they differ in number and/or type of arguments, is extremely useful and fairly simple to realize. The general result type is slightly more difficult to realize but gains a lot combined with an operator concept, the latter being simple to realize. The next extension necessary for numerical computations but a little bit more difficult to implement are dynamic arrays.

Those extensions have been implemented in the language TPX - Turbo Pascal eXtended [Hu91], a superset of Turbo Pascal which is transformed into Turbo Pascal by a precompiler. The latter has been written using the compiler generating tools Rex and Lalr [Gro87, Vie89]. The precompiler turns out to be very fast. It comprises of libraries for vectors and matrices over real and complex numbers as well as intervals over those, a long real and interval arithmetic and much more.

A user friendly software environment for scientific computation should avoid as much redundant information in a program as possible. Traditional programming requires type declarations etc. in order to produce fast code. For production code this is worth striving for. For development or research of algorithms it hinders. Therefore we are aiming on a software environment accepting the mathematical notation where possible without overhead.

There are systems like Matlab going far in this direction for matrix computations. However, they are fairly inflexible. From a mathematical point of view it should be possible to define matrices over some basetype together with the usual operations. The basetype, however, could be the set of real numbers als well as complex intervals or matrices again (for block matrices). Moreover, special structures like sparsity or symmetry should be possible together with efficient implementations.

This is possible, up to a certain point, using object oriented programming languages like C++. If the same is done with more comfort to the user by an interpreting system this has usually to be paid by a fairly poor performance. Methods will be presented to overcome this difficulty. It will be incorporated in the system ABACUS, a prototype of which we hope to be able to demonstrate.

Bibliography

[ACR86] ACRITH High-Accuracy Arithmetic Subroutine Library: General Information Manual, IBM Publications, GC33-6163 (1985)

[Ar86] ARITHMOS, Benutzerhandbuch, Siemens AG, Bibl.-Nr. U 2900-I-Z87-1 (1986)

[ArDeDu89] Arioli, M., Demmel, J.W., Duff, I.S.: Solving Sparse Linear Systems with Backward Error, SIAM J. Matrix Anal. Appl. 10, No. 2, 165 - 190 (1989)

[Bel75] Bellmann, R.: Adaptive Control Processes, Princeton University Press (1975)

[Gro87] Grosch, J.: Rex - A Scanner Generator, Report Nr. 5, GMD (1987)

[Hu91] Husung, D.: TPX Precompiler für Turbo Pascal, Bericht 91.1 des Forschungsschwerpunktes Informations- und Kommunikationstechnik der TUHH, 1991

[Ja85] Jansson, C.: Zur linearen Optimierung mit unscharfen Daten, Dissertation, Kaiserslautern (1985)

[JaRu90] Jansson, C., Rump, S.M.: Rigorous Solution of Linear Programming

Problems with Uncertain Data, to appear

[Ku76] Kulisch, U.: Grundlagen des numerischen Rechnens (Reihe Informatik, 19), Mannheim-Wien-Zürich, Bibliographisches Institut (1976)

[Ku81] Kulisch, U.W.: Computer Arithmetic in Theory and Practice, Academic Press, New York (1981)

[Ru80] Rump, S.M.: Kleine Fehlerschranken bei Matrixproblemen, Dissertation, Universität Karlsruhe (1980)

[Ru83] Rump, S.M.: Solving Algebraic Problems with High Accuracy, Habilitationsschrift, in: A New Approach to Scientific Computation, Hrsg. U.W. Kulisch und W.L. Miranker, Academic Press, 51 - 120 (1983)

[Ru90] Rump, S.M.: Rigorous Sensitivity Analysis for Systems of Linear and Nonlinear Equations, MATH. of Comp., Vol. 54, Nr. 10, 721 - 736 (1990)

[Vie89] Vielsack, B.: Lalr for LALR(1) - Grammers, Reports of the GMD (1989)

Polyalgorithms with Automatic Method Selection

for the Iterative Solution of Linear Equations and Eigenproblems

W. Schönauer, R. Weiss, P. Sternecker
Rechenzentrum der Universität Karlsruhe
Postfach 6980, D-7500 Karlsruhe 1, Germany

Abstract

In the kernel of a black-box PDE solver, like FIDISOL, there is an iterative linear solver, like LINSOL, that must be robust and efficient. Up to now no single method has the desired properties. We use as a compromise a polyalgorithm that is composed from several CG-type methods, switching from one method to the other, if necessary. The corresponding strategy and investigations to improve the properties are discussed. Briefly we report about similar investigations for the general symmetric eigenvalue problem.

1. Introduction

With the advent of ever larger supercomputers and the progress in algorithm and software development, there is an increasing demand for "black-box" type solvers for PDEs (partial differential equations). For structural analysis there are many FEM (finite element method) program packages that solve a distinct variational equation with great flexibility of the geometric conditions. Here the operator and the solution method are fixed, and thus the resulting sparse linear system that must be solved in the kernel of the solution algorithm has usually well-known properties.

The situation is quite different for more general black-box solvers. In the FIDISOL program package, that has been developed in our research group, see section 17 in [1], for elliptic PDEs a differential operator of the type

$$Pu \equiv P\,(x, y, z, u, u_x, u_y, u_z, u_{xx}, u_{yy}, u_{zz}) = 0 \qquad (1.1)$$

can be used, where u and P have m components for a system of m PDEs, and P is an arbitrary nonlinear function of its arguments. A similar operator Gu = 0 for the BCs (boundary conditions) on a rectangular domain is admitted. For the solution a variable order/variable step size FDM (finite difference method) is used. The consistency orders in the space directions can be prescribed by the user or can be selfadapted by the solution procedure to meet a given relative tolerance of the error. Thus neither the properties of the PDE and BC operators nor the form of the difference star are known to the designer of the program package. As a consequence the properties of the resulting linear system for the computation of the Newton-correction or of the error estimate are not known and may cover an extremely wide range. The solution of linear systems under those extreme conditons is the main subject of this paper.

For the solution of the extremely large and sparse linear systems (above all if they result from the discretization of 3-D problems) only iterative methods can be used. Usually the problem size is limited by the size of the matrix of the linear system, thus this matrix will use the whole available storage. For such types of problems the fill-in of direct methods would lead to prohibitive storage and computation requirements. The most efficient method for the iterative solution of certain types of linear systems are MG (multigrid) methods [2]. But MG is a whole family of methods that must be tuned individually for each problem to obtain the high efficiency. Thus MG is highly efficient, but not robust enough to be used as a type of black-box linear solver for the wide range of problems that result from a PDE solver like FIDISOL. Therefore we decided to use CG (conjugate gradient) type methods. But also CG is a whole family of methods with different properties. When we investigated these methods we recognized that methods that are highly efficient in one case failed completely in the other case. These investigations then finally led us to the polyalgorithm that is discussed below.

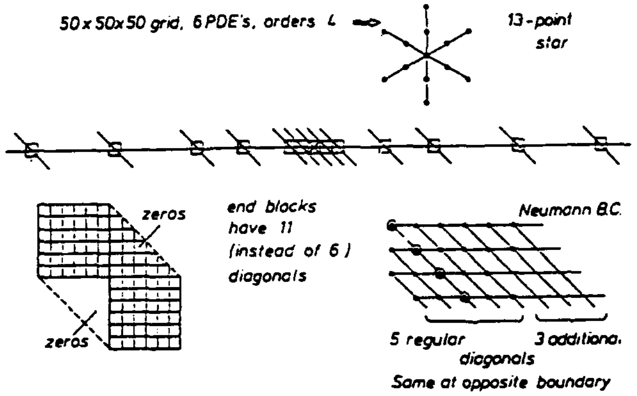

50x50x50 grid, 6 PDE's, orders 4
13-point star
zeros
end blocks have 11 (instead of 6) diagonals
Neumann B.C.
zeros
5 regular diagonals
3 additiona.
Same at opposite boundary

Fig. 1.1 Structure of the block diagonals for a system of 6 PDEs.

Before we discuss CG type methods we want to discuss how to store efficiently such large and sparse matrices that result from (high order) FDMs. This question is closely related to the key operation of all CG methods, namely the MVM (matrix-vector multiplication). Depending on the method, for each iteration step one MVM with the matrix A or also an additional MVM with the transpose A^T must be executed. In Fig. 1.1 the structure of the matrix for a 3-D system of $m = 6$ PDEs is depicted for consistency order 4 in each space direction. There result 13 "regular" block diagonals, but close to the wall and for Neumann BCs at the wall one-sided difference formulas with one additional point (to maintain the order) must be used, creating block-diagonals with only a few nonzero elements near border variables. After many discussions and investigations we decided to store such types of matrices by diagonals ("true" diagonals, not block-diagonals). But storing of true diagonals introduces a storage overhead for the storing of the zeros, see Fig. 1.1. The alternative is packed storing with 64 bits for the matrix elements and 32 bits for the indices in the diagonal. A storage-optimal decision for this situation is to store diagonals with more than 2/3 zeros as full diagonals (with zeros) and the other diagonals as packed diagonals.

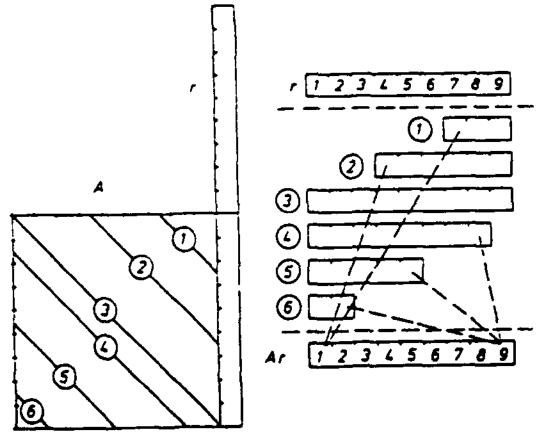


Fig. 1.2 Illustration of the MVM for a 9 x 9 matrix with 6 diagonals.

The MVM can be formulated easily by diagonals, see the illustration in Fig. 1.2. The diagonals of the matrix A are arranged for the multiplication with the elements of r and the dashed lines indicate the addition of the products to the result vector $d = Ar$. Note the different rules for upper and lower diagonals. For the multiplication $e = A^Tr$ the matrix A must not be transposed, but only the rules for upper and lower diagonals must be inverted. The tuning of this MVM for register vector computers is reported in [3].

2. Generalized CG methods

The iterative solution of extremely large and sparse linear systems is perhaps the most pressing problem in scientific computing today, several recent conferences were dedicated to this subject [4 -7]. As mentioned above we decided to use generalized CG methods for non-symmetric linear systems.

The linear system of n equations with non-singular n x n matrix A' that we want to solve is

$$A'x = b'. \tag{2.1}$$

Iterative solution means iterative reduction of the residual. If in (2.1) one equation is multiplied by 10^6 this does not change the solution, but changes completely the iteration process because now the residual of this equation gets the 10^6-fold weight and the iteration process will try to fulfil above all this equation. Therefore an appropriate normalization is absolutely necessary for CG-type methods. With $A' = (a'_{i,k})$ and the diagonal matrix $D = (d_i)$ we normalize (2.1)

$$DA'x = Db' \rightarrow Ax = b \tag{2.2}$$

with

$$d_i = (1/\sum_k |a'_{i,k}|)\text{sign } a'_{i,i}. \tag{2.3}$$

This means normalization to absolute row-sum equal to one and positive main diagonal and could be denoted as a simple preconditioning of (2.1) from the left.

Thus the normalized linear system with general (non-symmetric) matrix A to be solved and its residual r_k for an iterate x_k are

$$Ax = b, \quad r_k = Ax_k - b. \tag{2.4}$$

Experience tells us that the residual r_k oscillates heavily for generalized CG methods. Therefore we use a smoothing algorithm and compute parallel a to the iterated sequences r_k, x_k smoothed sequences $r_{min,k}$, $x_{min,k}$ as a weighted combination of the new iterated values and preceding smoothed values:

$$\begin{aligned} r_{min,\,k+1} &= a_k r_{min,k} + (1-a_k)r_{k+1}, \\ x_{min,k+1} &= a_k x_{min,k} + (1-a_k)x_{k+1}. \end{aligned} \tag{2.5}$$

The coefficient a_k is determined from $r^2_{min,k+1} = \min$, see [1], p. 261. There is no feedback of the smoothed sequences to the iterated sequences except at eventual restart points when we restart from the smoothed sequences. In the following we present the basic three methods of our ployalgorithm.

A quite simple access to generalized CG methods is the following. We define a pseudo-residual $\bar{r}_{k+1}$ (that is proportional to the true residual, see [1], p. 413) by

$$\bar{r}_{k+1} = Ad_k + \alpha_{1,k}r_k + \alpha_{2,k}r_{k-1} + \ldots + \alpha_{p,k}r_{k-p+1}, \tag{2.6}$$

i.e. as a combination of Ad_k, where d_k is a search direction, and p preceding (true) residuals. With

$$r_{k+1} = f\,\bar{r}_{k+1}, \quad f_k = 1 / \sum_{i=1}^{p} \alpha_{i,k} \tag{2.7}$$

follows

$$x_{k+1} = f_k(d_k + \alpha_{1,k}x_k + \alpha_{2,k}x_{k-1} + \ldots + \alpha_{p,k}x_{k-p+1}). \tag{2.8}$$

The coefficients $\alpha_{i,k}$ are determined from

$$\overline{r}^{2}_{k+1} = \min \rightarrow \alpha_{i,k} = - \frac{r_{k-i+1}\ Ad_k}{r^{2}_{k-i+1}} \tag{2.9}$$

From this set of formulas we get the PRES20 method by selecting $d_k = r_k$ and $p = 5$, this corresponds to a truncated ORTHORES method, see [8], p. 346. This method would not work in most applications. What makes this a useful method is a restart after 20 iterations from the smoothed sequence. This method is called PRES20. The value of $p = 5$ and the restart index 20 have been optimized by "numerical engineering" [9]. We start with $x_0 = 0$ or some guess x_0, $r_0 = Ax_0 - b$, and (also after each restart) with $p = 1, 2, 3, 4, 5$. For "sufficient" diagonal dominance PRES20 is the fastest method. PRES20 needs one MVM per iteration step.

For the search direction $d_k = A^T r_k$ we get the ATPRES method. This can be considered as a preconditioning of PRES from the left by A^T. Then the "iteration matrix" is AA^T. From inherent orthogonalities follows $\alpha_{i,k} = 0$ for $i > 2$, the method is an "exact" method that would terminate after n iterations in the absence of rounding errors. The multiplication $AA^T r_k$ is executed as $y_k = A^T r_k$, $z_k = Ay_k$ which is much more economic than a direct calculation via a matrix AA^T. Thus we need two MVMs per iteration step. ATPRES turns out to be a rather slowly convergent but at the same time very robust method.

The biconjugate gradient method BICO can be obtained from the following approach for the pseudo-residuals $\overline{r}_{k+1}$ and $\overline{r}^{*}_{k+1} = Ay_k - b$ (see [1], p. 257):

$$\begin{aligned} \overline{r}_{k+1} &= A\overline{r}_k + \alpha_k\overline{r}_k + \beta_k\overline{r}_{k-1} + \gamma_k\,\overline{r}_{k-2} + \ldots, \\ \overline{r}^{*}_{k+1} &= A^T\overline{r}^{*}_k + \alpha_k\overline{r}^{*}_k + \beta_k\overline{r}^{*}_{k-1} + \gamma_k\overline{r}^{*}_{k-2} + \ldots\ . \end{aligned} \tag{2.10}$$

The coefficients α_k, β_k are obtained from $\overline{r}^{T}_{k+1}, \overline{r}^{*}_{k+1}$ = extremum. From inherent orthogonalities follows $\gamma_k = \ldots. = 0$. With $\overline{r}_k = q_k r_k$ and the recursion $q_{k+1} = \alpha_k q_k + \beta_k q_{k-a}$ follows the relation

$$x_{k+1} = (\overline{r}_{k+1} + \alpha_k q_k x_k + \beta_k q_{k-1} x_{k-1})\ / q_{k+1}\ . \tag{2.11}$$

The auxiliary variable y_a is of no interest and does not appear in the algorithm. The algorithm can be reformulated in true residuals and auxiliary vectors [10]:

$$\begin{aligned} &r_{k+1} = r_k + a_k A\ p_k\ , \ r^{*}_{k+1} = r^{*}_k + a_k A^T p_k\ , \\ &p_{k+1} = r_{k+1} + b_k p_k\ , \ p^{*}_{k+1} = r^{*}_{k+1}\ b_k p^{*}_k\ , \\ &a_k = - \frac{r^T_k r^{*}_k}{(Ap_k)^T p^{*}_k}\ , \quad b_k = \frac{r^T_{k+1} r^{*}_{k+1}}{r^T_k r^{*}_k}\ , \\ &x_{k+1} = x_k + a_k p_k\ . \end{aligned} \tag{2.12}$$

We start with $r_0 = r_0^{*} = p_0 = p_0^{*} = Ax_0 - b$. The method is an "exact" one that terminates in the absence of rounding errors after n iterations with the exact solution. We need two MVMs per iteration step, namely Ap_k and $A^T p_k^{*}$. A breakdown is cured by a restart from the smoothed sequence if one of the denominators of a_k or b_k in (2.12) becomes (nearly) zero. BICO turns out to be a "medium" method just in between the two other methods: it is more robust than PRES20 and faster than ATPRES where it still works.

A unified theory for generalized CG methods has been presented by Weiss [11, 12]. The main results of this theory for the above discussed methods are: All generalized CG methods construct residuals that are in the Krylov space spanned by the iteration matrix and the initial residual. As the iteration matrices and strategies are quite different, also the convergence behavior is quite different.

For PRES20 we have the iteration matrix A and thus the convergence behavior depends on the eigenvalue distribution of A. The method converges if the symmetric part of A is positive definite [11]. The method is robust with respect to rounding errors because of the restarts.

BICO converges even if the symmetric part of A is not positive definite. As an "exact" method it is sensitive to rounding errors. The convergence is dominated by the eigenvalues of A. It is important for BICO to smooth the oscillating behavior of the original sequence (without feedback).

ATPRES uses the iteration matrix AA^T and thus the convergence behavior is dominated by the eigenvalues of this matrix. It has been shown in [11] that ATPRES minimizes the error in the Euclidean norm.

3. The polyalgorithm

In the LINSOL part of FIDISOL the three above mentioned generalized CG methods and several other iterative methods (see section 17 in [1]) have been included. For the solution of a PDE problem the user of the FIDISOL black box can select himself the iterative solver that he thinks, eventually by experience from preceding problems, to be the best method. But clearly it would be best that LINSOL selects itself the most efficient iteration method. Thus we need an "intelligent" linear solver.

It is also clear that we may not try to compute the eigenvalue distribution or the condition number of the iteration matrix A and then to decide which method is the best one, because the computation of these data is more time-consuming than the solution of the linear system itself. The solution x of the linear system (2.4) is a Newton correction for which no guess can be provided. The solution vector u of the PDE (1.1) enters itself into the matrix A and thus A may change its properties completely during the different steps of the Newton iteration process so that the decision for the type of linear solver must be made individually for each Newton step. How can we select the optimal iteration method under these circumstances?

Our experience with the three generalized CG-type methods showed that PRES20 was best for strongly elliptic problems, BICO was best for "medium" elliptic problems and ATPRES finally converged slowly also for weakly elliptic problems. So we composed from these three methods a polyalgorithm in the following way, which is a typical "numerical engineering" decision that makes use of "experience": The polyalgorithm starts with PRES20 and controls every 20 iterations at the restart point if the L_2-norm of the residual of (2.4), i.e. of the normalized equation $Ax = b$, has decreased by a factor of 0.5. If this is not the case, we switch to BICO. Then we check after every 1000 iteration steps if the residual has decreased at least by a factor of 0.1 over the last 1000 iterations. If this does not hold we switch to the "emergency exit" ATPRES. The reason why we check only after 1000 iterations is that BICO often converges in "steps" where it "falls down the cliff" at a certain iteration stage after a seemingly inefficient period. This strategy means that even in a single Newton step the iteration method of the linear solver may be changed.

In order to demonstrate how this polyalgorithm works we execute it for the solution of the following system of three nonlinear PDEs:

$$\begin{aligned} u_{xx} + u_{yy} + u_{zz} + u + R(uu_x + vu_y + wu_z) - h_1 &= 0, \\ v_{xx} + v_{yy} + v_{zz} + v + R(uv_x + vv_y + wv_z) - h_2 &= 0, \\ w_{xx} + w_{yy} + w_{zz} + w + R(uw_x + vw_y + ww_z) - h_3 &= 0. \end{aligned} \tag{3.1}$$

The h_i are determined that the solution is

$$u = \sin\alpha \cos\beta \cos\gamma,\ v = \cos\alpha \sin\beta \cos\gamma,\ w = \cos\alpha \cos\beta \sin\gamma \tag{3.2}$$

with $\alpha = 2\pi x$, $\beta = 2\pi y$, $\gamma = 2\pi z$. We prescribe Dirichlet boundary conditons, i.e. u, v, w on the unit cube. This example is a coarse model of the Navier-Stokes equations. The parameter R has the character of a Reynolds number. For small R we have strong ellipticity, for large R we have a more hyperbolic character. Thus we can change the type of problem by the value of R.

We generate A' and b' of (2.1) by FIDISOL for the consistency orders 4 in x, y, z-direction on a grid 40 x 40 x 40 (n = 192 000). The initial values for u, v, w are mean values of interpolations between the boundary values in x, y, z-direction. For these values the matrix A'of the first Newton step is used as test-matrix. We force LINSOL to reduce the L_2-norm of the relative residual $\|r_k\|/\|r_0\|$ of the normalized equations (2.4) to be reduced to 10^{-20}. In Fig. 3.1 we depict the log of the relative residual as a function of the MVMs. Note that PRES20 needs one MVM, but BICO and ATPRES need two MVMs per iteration step. The MVMs are a good measure for the computation. The parameter R is indicated as r on the top of the figures. For R = 1 and R = 10 the polyalgorithm follows PRES20, for R = 100 the polyalgorithm is between PRES20 and BICO, for R = 1000 the polyalgorithm follows BICO and for R = 10000 it follows nearly ATPRES. Thus the polyalgorithm with the above mentioned strategy follows nearly the best method. This is what we can call making "intelligent" software by numerical engineering.

4. Preconditioning

If we look in Fig. 3.1 at the convergence behavior for R = 1000 or R = 10000 we see a slow or very slow convergence. The dream of all users of CG methods is to have a preconditioning matrix that transforms the linear system to such a one that CG converges fast. If we put in the approach (2.6) the unknown search direction d_k as

$$d_k = P_k r_k \tag{4.1}$$

with a preconditioning matrix P_k (this is preconditioning from the right, in contrast to multiplication of the whole equation (2.1) by a preconditoning matrix form the left), we may ask: What is the optimal P_k? If $P_k = A^{-1}$ we have in (2.6)

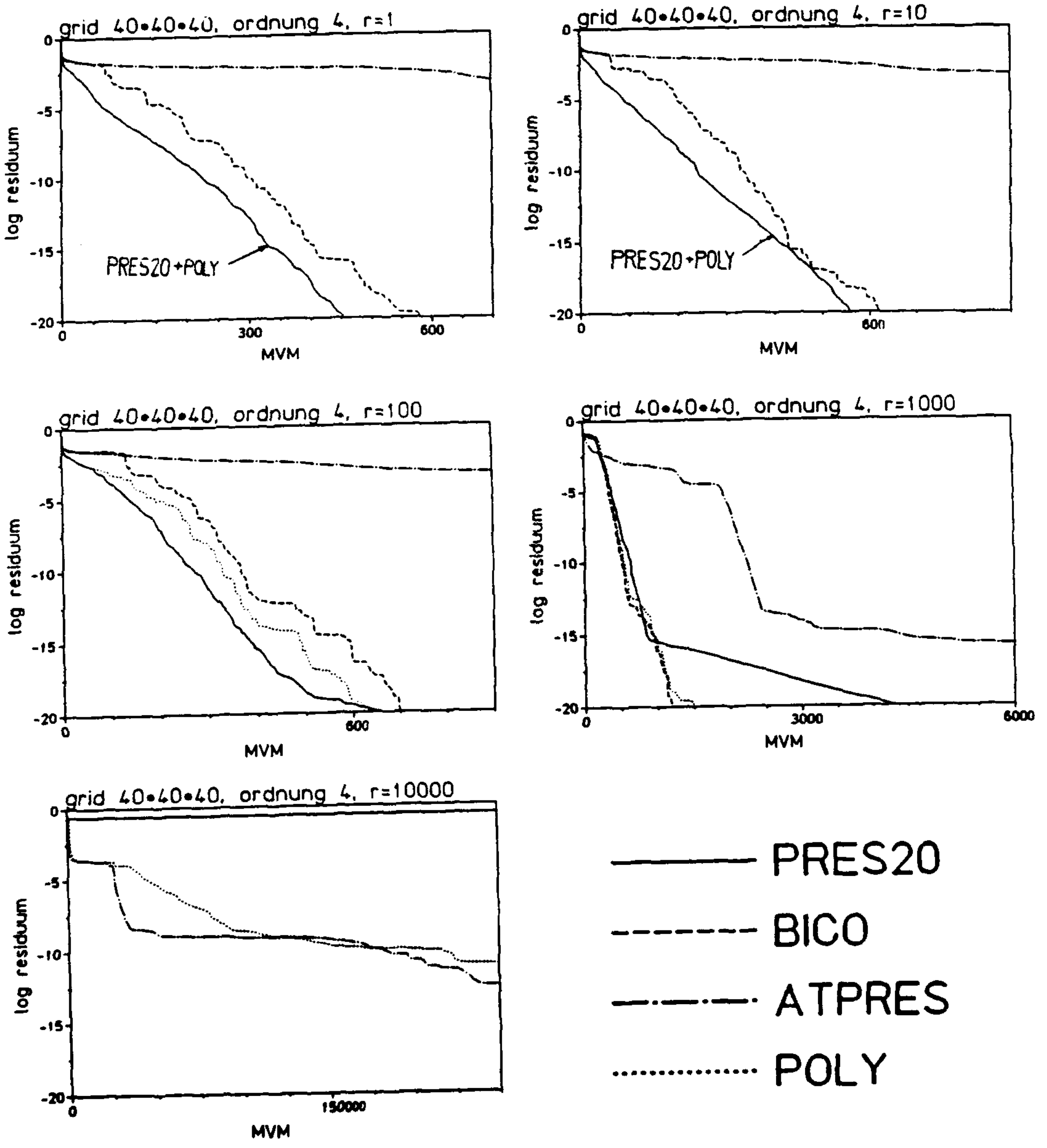


Fig. 3.1 Reduction of the relative residual for different algorithms over the number of matrix-vector multiplications (MVM).

$$Ad_k = AP_kr_k = AA^{-1}r_k = r_k. \qquad (4.2)$$

Then by $\alpha_{1,k} = -1$ and $\alpha_{i,k} = 0$ for $i > 1$ we have $r_{k+1} = 0$, thus r_{k+1} and $x_{k+1} = x$. Unfortunately this perception is of no practical use because determining d_k from $Ad_k = r_k$ is the same problem like the solution of (2.1) itself, and if we knew $P_k = A^{-1}$ we needed not to solve (2.1).

Before we present two approaches to preconditioning, we want to discuss the meaning of preconditioning from a heuristic point of view that may help us to understand precondtioning and thus to develop better preconditioners. The iterative solution of linear systems means iterative reduction of the residuals of all n equations. This reduction has two components: The reduction of the residual of each equation that couples a certain number of unknowns and represents a "local" transfer of information, and the global residual reduction that represents a "global" information transfer over all unknowns. All iteration methods reduce relatively fast the local residual to a certain level, but then the global residual reduction proceeds rather slowly because now the information of all equations must be balanced and the global information must be transferred via the local equations. Formally we would say that the condition number becomes larger if we increase the size of a linear system of the same structure, e.g. we increase the number of grid points in a FDM problem. Therefore the effect of a good preconditioner is to transfer quickly global information through the whole system. This is ultimately the reason for the high efficiency of MG. By the coarser grids variables with larger "distance" are coupled directly. Unfortunately the method is not robust enough for a black box solver like FIDISOL. Thus the design goal of a preconditioner must be to transfer global information in a robust and efficient way into the iteration process.

But there is an additional problem if we want to use supercomputers: An efficient use is only possible with optimal data structures that support the data flow through the pipelines. For this reason we decided to store the extremely large and sparse matrices by (eventually) packed diagonals. Thus a preconditioner must fit to this data structure. A well-known method for preconditioning is ILU (incomplete LU-decomposition)-preconditioning. We execute a Gauss elimination for the sparse matrix and drop all fill-in that would generate a nonzero element where we have a zero in the original matrix (= incomplete). In [13], p. 78 ff, we report about ILU-PRES20 and ILU-BICO where we solve approximately the preconditioning equation (4.2) $Ad_k = r_k$ by incomplete Gauss. This means a recursive procedure for the forward elimination that occurs once, but the recursive backward substitution occurs in each iteration step and destroys the efficient vectorization. In scalar mode the ILU-procedures are faster, but in vector mode they are slower than the basic methods. This means that in this case (note that we have arbitrary consistency order) the global information transfer is cheaper by the less efficient but efficiently vectorizable basic methods than by the ILU-methods that need far less iterations but more computation time.

Presently we are experimenting with an incomplete inverse computed by incomplete Gauss-Jordan. With an incomplete inverse we have in each iteration step for the approximate solution of (2.4) $Ad_k = r_k$ an additional MVM and thus no recursion, but the net effect is up to now dowtful. If we succeed in designing an efficient preconditioner this can be included as well in the polyalgorithm.

The other type of preconditioner that we call DARE (data reduction)-method [14], tries to combine ideas of multi-level methods (like MG) with CG methods. Because of time and space restrictions we present here only the basic ideas. In order to couple distant variables we "drop" on the grid lines of one space direction every second variable by replacing it with a cubic polynomial of the 4 neighboring remaining variables (of

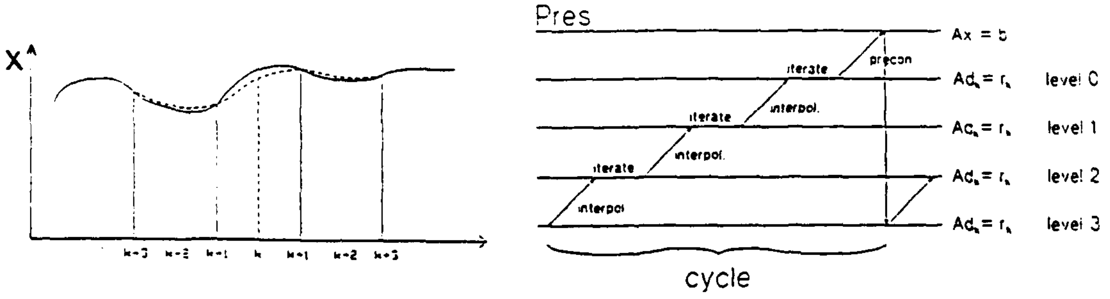


Fig. 4.1 Left: Cubic approach, right: DARE procedure.

the same component for a system of PDEs), see Fig. 4.1, left. Correspondingly we drop every second equation. Thus we have half the number of unknowns, defining a new reduction level. For 2-D or 3-D problems we reduce cyclically in x, y or x, y, z directions until we have only a few equations. The set of reduced matrices needs roughly the same storage as the original matrix. Then the DARE procedure is as follows, see Fig. 4.1, right: We execute a "leading" PRES20 method that has a residual r_k. Then we solve the preconditioning equation (4.2) $Ad_k = r_k$ on the lowest level, interpolate the dropped variables for the next level, iterate some steps and "climb upwards" in the same way until the leading PRES20 where we execute with this d_k the next iteration step, completing one cycle. Now the "polyalgorithm" is executed for the iterative solution on each level ("iterate" in Fig. 4.1, right.). There we switch from PRES20 to ATPRES (DARE1) or in another version from PRES20 to BICO to ATPRES (DARE2). The DARE procedure needs a really sophisticated control procedure for the switching and for the stopping of the iterations on each level, for details see [14].

In the upper part of Table 4.1 we present the computation time for the complete solution of the 3-D test-PDE (3.1) on a grid 40 x 40 x 40 with consistency orders 4, including the computation of error estimates, starting from an interpolation of the boundary values. There are 192 000 unknowns. In the lower part we present the same values for the 2-D test problem that results from the 3-D problem by dropping the z-dependency and the 3rd PDE. For this 2-D case we use a 150 x 150 grid with consistency orders 4. We have 45 000 unknowns.

Table 4.1: Timings and MVM for the 3-D and 2-D test-PDE on a Siemens S600 (Fujitsu VP 2600).

	R	PRES20	BICO	ATPRES	POLY	DARE1	DARE2
3-D	10	5.3 sec 192 MVM	6.1 sec 418 MVM	44.2 sec 4596 MVM	5.0 sec 192 MVM	4.6 sec	4.6 sec
	100	47.3 sec 3128 MVM	17.6 sec 1496 MVM	75.1 sec 8138 MVM	20.6 sec 1448 MVM	97.6 sec	19.9 sec
2-D	10	3.5 sec 1057 MVM	3.0 sec 1566 MVM	77.1 sec 48 888 MVM	2.8 sec 1306 MVM	1.3 sec	1.7 sec
	100	5.3 sec 1691 MVM	5.5 sec 3124 MVM	83.5 sec 53 620 MVM	5.3 sec 2722 MVM	2.4 sec	2.3 sec

In Table 4.1 we compare for R = 10, 100 the timings for the different methods. The polyalgorithm is always close to the best method. For 3-D at R = 10 DARE is just at the breakeven point and would pay only for more than 40 grid lines in each space direction, for R = 100 we are below the breakeven point. For 2-D DARE with 150 grid lines we are above the breakeven point and DARE pays. In Table 4.2 we compare the polyalgorithm and DARE for different values of R with roughly the same conclusions. In 3-D the case R = 1000 is obviously a rather critical problem and cannot be solved with the standard parameters of FIDISOL (we get a solution for other parameters). For R = 10 000 we have large error estimates.

Table 4.2: Timings for the 3-D and 2-D test-PDE on a Siemens S600 (Fujitsu VP2600).

	R	1	10	100	1000	10 000
3-D	POLY	5.0 sec	5.9 sec	20.6 sec	(*)	≻ 15 min
	DARE 1	4.6 sec	6.0 sec	97.6 sec	(*)	9.9 sec
	DARE 2	4.6 sec	6.0 sec	19.9 sec	≻ 15 min	48.1 sec
2-D	POLY	2.8 sec	3.8 sec	5.3 sec	6.5 sec	8.7 sec
	DARE 1	1.3 sec	1.5 sec	2.4 sec	2.9 sec	164 sec
	DARE 2	1.7 sec	2.1 sec	2.3 sec	11.3 sec	(*)

(*) divergence with the standard parameters of FIDISOL

5. The symmetric general eigenvalue problem

In our research group a FEM kernel program package VECFEM (vectorized FEM) with optimal data structures for vector computers has been developed [15]. Part of this project is the program FEMEPS (FEM eigenproblem solver) for the computation of the extreme eigenvalues and eigenvectors of large sparse symmetric general eigenproblems of type

$$Ax = \lambda Bx. \tag{5.1}$$

For details and also for corresponding references see [16]. Here we discuss only the polyalgorithmic properties. The symmetric matrices A and B are stored by (packed) diagonals.

The eigenvectors of the problem (5.1) are the stationary points x of the Rayleigh quotient

$$R(z) = (z^T Az)/(z^T Bz),\ z \neq 0. \tag{5.2}$$

We have $R(x) = \lambda$. As basic algorithm for the numerical solution we choose a Rayleigh quotient minimizing/maximizing technique. The algorithm is based on a local (with respect to the Hessean matrix of R) conjugate gradient like method coupled with the mini/maximization of R. Usually the convergence is rather slow. Therefore we use preconditioning by a few steps of the "classical" CG method, thus approximating a preconditioning by A^{-1}. The whole algorithm is smoothed by a relation like (2.5). For the computation of later eigenvectors, previous eigenvectors are eliminated by orthogonalization. Nevertheless the method may "fall asleep".

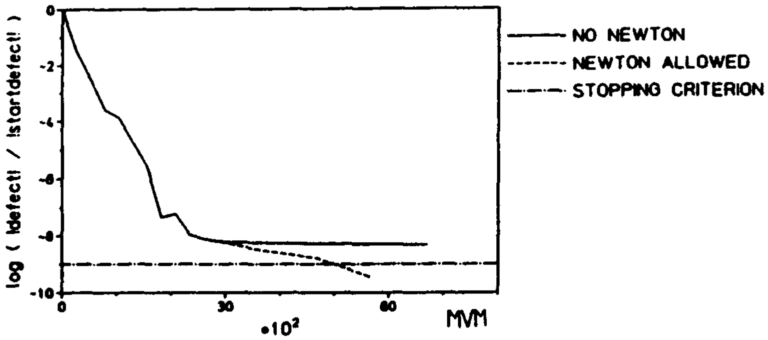


Fig. 5.1 Reduction of the relative residual of (5.1).

In this situation we use a polyalgorithm and switch to a projected Newton-method for (5.1) (with λ from R). Unfortunately the Newton matrix is nearly singular near eigenvectors and so the computation of the Newton corrections by the CG method has poor convergence. Therefore we execute only a few iterations of CG and use a "damped" Newton with an optimized relaxation factor. For the example of a combination material of wood layers and a steel bar with 2000 degrees of freedom the reduction of the relative residual of (5.1) is depicted in Fig. 5.1 over the MVM for the determination of the 4th eigenvector. When the Rayleigh quotient method falls asleep, the Newton method brings down the residual to the requested stopping criterion. Thus again a polyalgorithm helps.

6. Concluding remarks

Only if we consider such a wide scale of linear equations like those created by the FIDISOL program package, we become aware how far we are still away from a satisfactory linear solver that is really robust and efficient. We believe that also in the future there will be no single method that is efficient for all types of problems. Therefore a black box linear solver must be a polyalgorithm with several methods. The problem then is, how to choose the optimal method.

In our polyalgorithm we offer a scale of generalized CG methods with increasing robustness but at the same time decreasing efficiency, this is the price to be paid for the robustness. If the convergence rate exceeds certain limits a switch to the next method is executed. For very large problems that arise with the advent of ever large supercomputers multilevel methods become a necessity. Here the polyalgorithm must be shifted to the iterations on the levels of the method. What we need are "intelligent" polyalgorithmic linear solvers.

A similar situation holds for the iterative solution of large eigenvalue problems. The efficient Rayleigh-Quotient method may fail. Therefore an emergency exit to a slow but rather robust projected Newton method leads to a polyalgorithm with an acceptable compromise between efficiency and robustness.

References

1. W. Schönauer, Scientific Computing on Vector Computers, North-Holland, Amsterdam 1987.
2. W. Hackbusch, U. Trottenberg (Eds.), Multigrid Methods, Lecture Notes in Mathematics 960, Springer, Berlin 1982.
3. R. Weiss, H. Häfner, W. Schönauer, Tuning the matrix-vector multiplication in diagonal form, in D.J. Evans et al. (Eds.), Parallel Computing '89, North-Holland, Amsterdam 1990, pp. 93 -98.
4. D. R. Kincaid, L.J. Hayes (Eds.), Iterative Methods for Large Linear Systems, Academic Press, Boston 1990.
5. D.L. Boley, D.G. Truhlar, Y. Saad, R.E. Wyatt, L.A. Collins (Eds.), Practical Iterative Methods for Large Scale Computations. Computer Physics Communication 53, numbers 1 - 3, May 1989, pp. 1 - 477.
6. O. Axelsson, L. Yu.Kolotilina (Eds.) Predonditioned Conjugate Gradient Methods, Lecture Notes in Mathematics 1457, Springer, Berlin 1990.
7. IMACS International Symposium on Iterative Methods in Linear Algebra, Brussels, Belgium, April 2 - 4, 1991, proceedings to appear.
8. L.A. Hageman, D.M. Young, Applied Iterative Methods, Academic Press, New York 1981.
9. W. Schönauer, H. Müller, E. Schnepf, Pseudo-residual type methods for the iterative solution of large linear systems on vector computers, in M. Feilmeier et al. (Eds.), Parallel Computing 85, North-Holland, Amsterdam 1986, pp. 193 - 198.
10. R. Fletcher, Conjugate gradient methods for indefinite systems, in G.A. Watson, Proc. of the Dundee Biennial Conference on Numerical Analysis, Springer, Berlin 1975, pp. 73 - 89.
11. R. Weiß, Convergence Behavior of Generalized Conjugate Gradient Methods, Dissertation Universität Karlsruhe 1990 and Interner Bericht Nr. 43/90 des Rechenzentrums der Universität Karlsruhe, 1990 (free copies can be obtained on request).
12. R. Weiß, Properties of generalized conjugate gradient methods, submitted to J. of Numerical Linear Algebra with Applications.
13. W. Schönauer, E. Schnepf, H. Müller, The FIDISOL Program Package, Interner Bericht Nr. 27/85 des Rechenzentrums der Universität Karlsruhe, 1985.
14. R. Weiß, W. Schönauer, Data reduction (DARE) preconditioning for generalized conjugate gradient

methods, in [6], pp. 137 - 153.

15. L. Groß, P. Sternecker, W. Schönauer, Optimal data structures for an efficient vectorized finite element code, in H. Burkhard (Ed.), CONPAR 90/VAPP IV, Lecture Notes in Computer Science 457, Springer, Berlin 1990, pp. 435 - 446.

16. P. Sternecker, L. Groß, W. Schönauer, A polyalgorithm for the solution of large symmetric general eigenproblems, in [7], to appear.

The Computer as Statistical Assistant

by

J.A. Nelder
Dept. of Mathematics
Imperial College
180 Queen's Gate
London SW7 2BZ
UK

Abstract

Types of front-ends for scientific software will be distinguished: decision trees, knowledge-enhancement systems, knowledge-enabling systems, and systems with higher levels of semantic help. Particular characteristics of statistical front ends are (1) concern with inference rather than diagnosis, (2) the high level of abstraction in the rules, and (3) three-way communication between the back end, the front end and the user. The paper will discuss the construction of front ends for statistical software, drawing on experiences in the development of GLIMPSE, a front end for GLIM, and subsequent developments arising from the the FOCUS project. The tools required if front ends are to evolve flexibly in response to new statistical techniques will be outlined.

0. Introduction

Statistics demonstrates many interesting aspects of programming environments for scientific work. It is also a subject in which substantial work has been done on the construction of software, including knowledge-based front ends (KBFEs). This paper (1) looks at some existing types of front ends, (2) discusses some particular characteristics of statistical front ends, (3) uses GLIMPSE, a KBFE for GLIM, and its successor FAST as an illustration of a particular front end, and (4) ends with a discussion of future developments.

1. Types of Front End

There are various classes of advisory systems with increasingly complex structures.

1.1 Decision Trees

At each node of the tree the user is asked for information and his answer determines the branch taken. At the bottom of each branch is a recipe for action. Note (i) the close relation to diagnostic keys, (ii) that not all structured information can be expressed conveniently by trees, and (iii) that the expertise in such systems lies in the construction of the tree.

1.2 Knowledge-Enhancement Systems (Hand)

In these the user in presented with material about some class of techniques, say multivariate analysis, structured in the form of a graph through which (s)he can browse; there may be facilities for reading at different depths, for getting references to further work, etc.. Note that there are no algorithms attached.

1.3 Knowledge-Enablement Systems

In these the previous type is augmented by algorithms which can carry out the techniques covered in the advice. Syntactic help will be given in setting up the parameters for the algorithm required.

1.4 Systems with Higher-Level Guidance

These give guidance not just on executing a single procedure, such as fitting a regression equation, but also on selecting the explanatory variables, checking the internal consistency of the fit, and so on. These will be exemplified by GLIMPSE in section 4.

2. Special Features of Statistical Advisory Systems

(i) Statistical systems deal with inference rather than diagnosis (compare medical systems). Statistical inference is both a more complicated and less well-defined process than diagnosis.

(ii) The rules in statistical systems are much more abstract than those in e.g. medical diagnosis systems. This is because statistical techniques can be applied in many diverse applications, hence the rules in a statistical KBFE must be sufficiently general to cover all these applications.

(iii) Many expert systems involve a two-way interaction only between the user and the system. In statistical systems there are both rules and an algorithmic engine, and the user communicates with both. This affects the design of any shell used in constructing the system. Many standard shells support only two-way interaction between user and expert system.

3. Basic Processes in Statistics

The contribution of statistics to a research programme involves three major components:

(i) Design: get maximum information on the quantities to be measured with the resources available.

(ii) Analysis: find relevant parsimonious models that account for the systematic and random parts in the data.

(iii) Inference: draw quantitative conclusions from the analysis.

Advisory systems may be designed for a single component, or all three in combination. GLIMPSE deals with (ii). I know of no system that covers all three components.

4. Designing the Interaction with the User

A major problem in designing an advisory system is to find ways of merging the system's knowledge, encoded as the rules of an abstract statistician, with the user's knowledge.

Information exists in

(i) The user's mind

(ii) The abstract statistician's rules

(iii) Output from the back end

Information may be processed by any of these sources and the results communicated to the others. The user interface must be designed to facilitate exchange of information along all relevant pathways. Questioning is a major activity.

4.1 The form of the interaction in GLIMPSE

In GLIMPSE the user can follow the abstract statistician's rules always, but (s)he does not have to accept its advice. This non-authoritarianism allows the user's knowledge to be fed into the system.

In GLIMPSE primitive steps in an analysis are called tasks. The abstract statistician's strategies are expressed as series of tasks. The user may execute any task or series of tasks without consulting the abstract statistician. This produces a transparent system, i.e. a glass-sided box rather than a black box. The user may construct new strategies if they are expressible using the tasks provided.

In GLIMPSE the user can respond to a question in various ways:

(i) What answers are possible?

(ii) What does a jargon word, e.g. 'deviance', mean?

(iii) What is the background to the question?

(iv) Why is the question being asked?

(v) What would the system suggest?

(vi) What was the question again?

It is very important that a user be able to back-track and give a revised answer to a previous question that was answered wrongly.

6. Components of GLIMPSE

6.1 The Task Language

GLIMPSE supports a task language, which is at a higher level than that of its back end, GLIM. A typical task is:

plotgraph ($\log(y)$ vs $\log(x)$)

check (y in-range $< 0, 100 >$ & $\int(y)$)

find changes in fit over 3.5

All tasks are translated into GLIM macros and sent to GLIM to be executed. The results are then displayed, perhaps after some post-processing. Tasks are the level-0 help in GLIMPSE, and form a knowledge-enablement system. On top of this two further levels of help are constructed, level-1 or specific help, and level-2 or strategic help. These higher levels of help are built round the idea of an activity in an analysis.

6.2 Activities in GLIMPSE

These are:

(i) *Data input.* The data are read or typed in.

(ii) *Data definition.* The user is asked about the properties of the variables read in under (i) and about the relations between them.

(iii) *Data validation.* The data values are checked against the definitions and any inconsistencies pointed out.

(iv) *Data exploration.* Properties of the response variable and relations between the response variable and each of the explanatory variables are explored with a view to defining a subclass of GLMs which it would be useful to explore formally.

(v) *Model selection.* Using the error distribution and link function suggested by (iv), the effects of the explanatory variables are tested by model fitting, and a set of parsimonious models is produced.

(vi) *Model checking.* A model produced by (v) can be checked by the use of a battery of model-checking techniques to establish its internal consistency.

Level-1 (specific) help is concerned with components of an activity, for example the relation between the response variable and a particular explanatory variable in data exploration. Level-2 (strategic) help gives advice on a whole activity. Thus for data exploration, level-2 help looks first at properties of the response variable itself, followed by its relation to each of the possible explanatory variables.

The hierarchical structure of the help can be continued by defining a level-3 help which, by providing advice for moving between activities, would allow help with an entire analysis. If then help is provided with both the design and inference components, the resulting level-4 help would cover a whole cycle (design - analysis - inference) of a research programme.

6.3 On-Line Documentation

This comprises text files dealing with the task language, and background information on each of the activities, the strategies employed, and the general and specific tasks available at that activity. A major problem in providing the background information is to decide on the appropriate technical level to assume. Something must be assumed, otherwise one is committed to writing a whole statistical textbook. Provision of different levels of explanation is a partial solution, but was not implemented in GLIMPSE, mainly because of lack of time.

6.4 The Lexicon

This allows the user to look up unfamiliar words that may be met during a session. The lexicon is a simple browser, with definitions at each node, and lists of words defined in associated nodes. Again there are problems of how far down the definitions should go.

6.5 The Abstract Statistician

This contains the statistical expertise of the system, encoded as Prolog rules. It uses an adaptation of an expert-system shell called APES (augmented Prolog for expert systems); APES supports query-the-user, which automatically queries the user for information required but not found in the rulebase, and provides explanations of how the user has arrived at a particular place in the strategy, and why a particular question is being asked. The rules in the abstract statistician were developed using the past experience of the statisticians in the project, and tested on a diverse collection of data sets that covered much of the range of GLMs. Like the rest of the front end this part was written in Prolog.

7. Future Prospects

The construction of advisory systems in statistics is still at an early stage. We have much to learn about (i) the development of strategies for statistical analysis, and (ii) the development of computing tools to produce systems that users will find easy and convenient.

7.1 The Development of Strategies

Current strategies arise from the experience of developers, and are tested by trying them out on a range of data sets. The process of knowledge acquisition by the system can follow at least three routes: (i) the developers contemplate their navels and produce rules, (ii) the developers employ a knowledge-acquisition specialist who questions them about their working procedures and from the results produces rules, which are then fed back to the developers for amendment, and (iii) the developers automate the process by sitting experts at the terminal with a some training sets of data, and a program induces their rules from their behaviour.

Route (iii) was used by Gale and Pregibon in the project Student. Note that different experts will produce different rule sets, and that it may impossible to fuse several sets of rules into a consistent whole. Also the process is very dependent on the choice of training sets. Routes (i) and (ii) are less dependent on the training sets, because the developers are drawing on their past experience in analysing data, and taking account of what they have learnt in the process. Training sets remain important, however, as a way of testing the rules once formulated. There can be no question of any formal proof that a set of rules will always work; almost certainly it will be possible to design pathological data sets for which any given rule-set will fail.

There is a danger that the strategies embedded in a successful advisory system will become frozen and inflexible, or will depend entirely on the originators for revision. If possible, this should be avoided.

7.2 The Evolution of Strategies

In GLIMPSE the user may execute a personal strategy provided that it can be expressed as a series of tasks already defined in the task language. However (s)he cannot store such such a strategy in a form which others can retrieve and use for themselves. To enable this to be done we need some extensions to the tools currently available.

First, the task language must be extended to include the standard notion of procedures, and to have the standard control structures for looping and branching. Secondly, because questioning is such a basic activity, the facilities for querying the user, which in GLIMPSE are part of the APES shell, must be brought up to a level where the developer of a new strategy can employ them in his procedures. (An analogous facility is included in the latest version of Genstat 5, and allows an interactive system to become a conversational one.) Thirdly, we need a way in which a strategy-developer can define the syntax and semantics of new tasks, so that new strategies are not restricted to the existing set of tasks.

None of these requirements makes excessive demands on current computing technology, though the third one in particular will need very careful design if it is to be straightforward for a developer to use.

If these facilities existed the way would be open for the evolution of strategies; new strategies could be defined, stored, retrieved by others, and compared with the existing standard. (To establish methods of comparison will not be easy!) This evolution would not be Darwinian, because it would not proceed by the occurrence of 'random' mutations followed by selection. New strategies would arise from new theory and new ideas by statisticians; the selection component would still be there, of course, and progress could well be fast. It should certainly be faster than Darwinian evolution.

7.3 Future Computing Environments

An example of what sort of computing environment the developer of an advisory system can expect in the near future is provided by the FOCUS project. This is being developed under the Esprit II programme of the EEC, with workers in the U.K., Belgium, Netherlands, Germany and Spain taking part. The project aims to develop a toolkit for the construction of front ends for scientific software. Included are both libraries (called open systems) and packages (called closed systems). The architecture which has been developed for FOCUS involves three separate Prolog systems which converse with each other by sending standardised messages. Components of the architecture include:

(i) The harness: this includes a presentation layer, which handles the interactions with the user on the screen, and dialogue control, which deals with the message passing.

(ii) The back-end manager, which handles the traffic between the front ends and the back ends.

(iii) The problem solver. This provides an inference engine, which includes a reason-maintenance system to ensure that the state of a system remains consistent when previous assertions are withdrawn; it can also provide explanations of failure.

In addition some general tools will be provided to help the constructor of a knowledge-based module (front end). One is

(iv) On-line documentation tool. This allows the mixing of text and pictures, combined with a hypertext-like framework for setting up reading paths for the user.

The back-end manager will allow the developer to access several back ends; one of which will be a data-manager for the storage, manipulation and retrieval of basic numerical data structures such as vectors and matrices. Another might be a graphics package for particular types of picture. Note that the components listed above need not all reside on the same machine; the architecture is designed to make it possible to use different components on different machines.

8. Conclusion

Statisticians have much work to do in developing large-scale strategies of design, analysis, and inference. This work will need to be done by those whose orientation is towards science, rather than mathematics.

Computing science has still to develop tool kits for the new generation of machines that will use their power while at the same time giving the developer an interface that (s)he can use without detailed knowledge of the internal structure.

Statisticians and computing scientists need to build close working relationships if useful statistical advisory systems are to appear.

Toward an Intelligent System for Mathematical Software Selection[*]

Ronald F. Boisvert[†]
Computing and Applied Mathematics Laboratory
National Institute of Standards and Technology
Gaithersburg, MD 20899

January 10, 1990

Abstract

A vast collection of mathematical and statistical software is now available for use by scientists and engineers in their modeling efforts. This software represents a significant source of mathematical expertise, created and maintained at considerable expense. Unfortunately, the heterogeneity of the collection makes it difficult simply to determine what software is available to solve a given problem. In mathematical problem-solving environments of the future such questions will be fielded by expert software advisory systems. In this paper we describe knowledge engineering techniques and associated selection heuristics which can be used to develop such systems. A prototype under development for the Guide to Available Mathematical Software project at the National Institute of Standards and Technology is demonstrated.

1 Introduction

Users of large scientific computing facilities often have substantial collections of mathematical and statistical problem-solving tools at their disposal. At the National Institute of Standards and Technology (NIST), for example, computer users may choose from some 5000 software modules contained in 40 packages. With seven computer systems available, this amounts to nearly 18000 module implementations. This bounty has placed a great burden on computer support staffs who wish to provide the most appropriate tools to their clients for a given task. In such an environment, it is a tedious and error-prone task simply to determine what software is available to solve a given problem on a given computing device. The difficulty lies both in the volume and heterogeneous nature of information describing the software and in the task of matching roughly defined problems with sharply defined software.

Users often turn to human experts for help with software/hardware selection. Usually, such advice is dispensed by computer center consultants or numerical analysts, although neither is particularly suited to this task. Computer consultants often do not have the mathematical knowledge to correctly match problems with software, and numerical analysts, while in possession of very deep knowledge in specific areas, do not have sufficient breadth of knowledge to field all queries. Neither have convenient access to software information. Initial user contacts are best handled by mathematical software generalists, i.e., consultants with (a) sufficient mathematical knowledge to

[*] Contribution of the National Institute of Standards and Technology, not subject to copyright in the United States.
[†] Electronic mail: boisvert@cam.nist.gov or rfb@nistcs2.bitnet

Table 1: Objects in the GAMS Knowledge Base

Object	Typical Properties	Count
Problem	Name, description, verbose description	783
Package	Name, description, type, portability, developer	40
Module	Name, description, how to use	5162
Computer	Name, location, operating system, compilers	8
Package Implementation	Version, access procedures, support level	88
Module Implementation	Precision, comment	17972

guide users from vague problem descriptions to more precise ones suitable for software selection, (b) ready access to up-to-date information about existing computing devices and the software available there, and (c) the ability to discriminate between software for solving similar problems based upon the user's individual needs. If satisfactory results are not obtained in such a consultation, then referral to a specialist is indicated.

It has been widely recognized that automated systems are quite well suited to play the role of mathematical software advisor, and many computer centers have developed systems of this type for their users. Several tools have been developed to aid in the construction of such systems [2], [8], [9]. In recent years there has been an interest in exploring how techniques of knowledge engineering and expert systems can be used to improve their performance [1], [5], [6], [7], [10], [11], [12].

In this paper we will describe knowledge engineering techniques and associated selection heuristics which can be used to develop systems which perform at the level of a mathematical software generalist. These techniques are being used in the further development of the GAMS Interactive Consultant, the advisory system associated with the NIST Guide to Available Mathematical Software (GAMS) project [2].

2 Knowledge Engineering for Software Selection

The GAMS knowledge base contains the data necessary to support various tasks associated with mathematical software selection. It is a collection of *objects* and *links* of various types. The object types define the kinds of objects in our knowledge base; all objects of a given type share a fixed set of properties which distinguish them from other objects of the same type. The link types define the ways in which objects are interrelated. Tables 1 and 2 summarize the basic object and link types. These are described in detail in [1], and their implementation in a particular database system is described in [4]. Table 1 also lists the number of each object in the current GAMS knowledge base.

2.1 Mathematical Knowledge

Problem objects in the GAMS knowledge base represent mathematical problem domains; the links show the inclusion of specific problem domains within more general ones. In effect, this defines a tree-structured taxonomy of mathematical problems. The GAMS problem taxonomy is quite detailed, containing about 783 problems extending to seven tree levels. The top-most level of the system specifies the broadest problem classes such as Linear Algebra, Differential and Integral Equations, and Statistics. An example of a complete sub-tree rooted at a particular problem is

Table 2: Links in the GAMS Knowledge Base

From	To	Implied relation
Problem	Problem	is an instance of
Module	Package	is part of
Module	Problem	applies to
Module	Module	is usually used with
Package implementation	Computer	is available on
Module implementation	Package implementation	is part of
Module implementation	Module implementation	is a version of

Table 3: Subtree F of the Problem Classification Scheme

F.	Solution of nonlinear equations
F1.	Single equation
F1a.	Polynomial
F1a1.	Real coefficients
F1a2.	Complex coefficients
F1b.	Nonpolynomial
F2.	System of equations
F3.	Service routines (e.g., check user-supplied derivatives)

given in Table 3; a complete listing may be found in [3].

The problem taxonomy can be used by advisory systems as a decision tree to lead users from very general to very precise problem statements. If one or more classifications are given to each software module, then the system can present users with a list of software for use in solving problems of any given class. The GAMS problem taxonomy has been used in this way by a number of advisory systems[1], including our own [1].

There are difficulties with this approach, however. Many mathematical problems are sufficiently common and well-understood that there are many software modules available for their solution. For example, the GAMS database contains 34 modules with classification F2 (Solution of a system of nonlinear equations). Users need further help in discriminating among these choices. If the problem taxonomy is the only method of navigation available to an advisory system, then the only way to reduce the number of modules per class is to make the problem taxonomy more detailed. In some cases this is feasible, but this does not provide a satisfactory general solution. Adding many levels of refinement to the problem taxonomy soon leads to an explosion of nodes, most of which have no software at all. Eventually, further refinement based upon mathematical problem alone is not possible. At this stage, the only difference between modules are properties of the algorithm used or the details of its implementation as computer software. We feel that this type of breakdown is inappropriate in a problem taxonomy.

The GAMS Interactive Consultant currently allows users to declare certain properties required of acceptable software modules. In particular, users need only see modules available on particular computer systems, in particular software packages, in a particular arithmetic precision, or in non-

[1] At Amoco Production Research, University of Texas Center for High Performance Computing, Lawrence Livermore National Laboratory, and the Konrad-Zuse-Zentrum für Informationstechnik Berlin, for example.

Table 4: Module Features for Class E1a

	Feature Type	Feature
1	type of problem	piecewise polynomial (PP) interpolation
2		osculatory PP interpolation
3		optimal interpolation (determine range of values)
4	visual property	as smooth as possible (spline)
5		preserve convexity
6		preserve monotonicity
7		Akima smoothing
8	polynomial degree	quadratic
9		cubic
10		quintic
11		user selected
12	end condition	natural (zero derivatives)
13		user-supplies derivatives
14		software estimates derivatives
15		not-a-knot (extra continuity near ends)
16		periodic
17		compatible with monotonicity
18	other feature	user supplies knots (B-spline breakpoints)

proprietary packages. Although using this information can sometimes significantly reduce the number of modules presented to the user, it is often not enough.

2.2 Extended problem/software knowledge

In order to further discriminate between modules in a given problem class an advisory system must have further information about the distinguishing properties of each module. These properties are necessarily problem-dependent, and can only be ascertained after a detailed study of all of the software modules in a given class. For example, consider the class E1a, Interpolation of univariate data by polynomial splines (piecewise polynomials). The GAMS database contains 53 modules with this classification, although only about 34 remain after eliminating those which are double precision versions of others. These are distinguished by characteristics such as the polynomial degree, the amount of smoothness, how end conditions are handled, and so on.

In general, after surveying the software in a given class, one discovers a set of features which can be partitioned into a small set of feature types. Thus, we extend our knowledge base by defining a new object, the *feature*. Each feature has a *feature type*, and is linked to a particular class. The features of class E1a are given in Table 4.

We next need a way to encode information about which features apply to each software module in a given class. To do this we associate an integer with each (module, class, feature) tuple that indicates the status of the feature for the given module in the given class. Possible values are

1	feature is always present
0	feature is optionally present
-1	module does not have this feature

The set of feature status indicators associated with a given (module, class) pair is called a *module feature vector*. The module feature vectors for class E1a are given in Table 5.

Table 5: Module Feature Vectors for Class E1a

		Feature Vector																	
Module	Package	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
A507	CALGO	-1	1	-1	-1	-1	1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
A514	CALGO	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
A547	CALGO	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	0	-1	0	-1	0	-1	-1
A574	CALGO	-1	1	-1	1	-1	-1	-1	1	1	1	-1	-1	-1	-1	-1	-1	-1	-1
A592	CALGO	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
A600	CALGO	-1	1	-1	-1	-1	1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
ICSCCU	IMSL	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
ICSICU	IMSL	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
ICSPLN	IMSL	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1
IQHSCU	IMSL	1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1
BSINT	IMSL/MATH	1	-1	-1	0	0	0	0	1	-1	-1	-1	-1	-1	-1	1	-1	-1	1
CSAKM	IMSL/MATH	1	-1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1
CSCON	IMSL/MATH	1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
CSDEC	IMSL/MATH	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	0	-1	0	-1	-1	-1
CSHER	IMSL/MATH	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
CSINT	IMSL/MATH	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
CSPER	IMSL/MATH	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1
QDDER	IMSL/MATH	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
QDVAL	IMSL/MATH	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
SPISC1	JCAM	-1	1	-1	0	0	0	0	1	1	1	-1	-1	-1	-1	1	-1	-1	-1
E01BAF	NAG	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
E01BEF	NAG	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	1	-1
E02BAF	NAG	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	1
PCHEZ	NMS	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
PCHEZ	NMS	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	1	-1
CSPFI	PORT	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
CSPIN	PORT	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
E1ACS	SCIDESK	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	0	0	1	-1	-1	-1
E1AM	SCIDESK	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	1	-1
SPLINE	SCRUNCH	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
BINT4	SLATEC	1	-1	-1	-1	1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	-1	-1
BINTK	SLATEC	1	-1	-1	0	0	0	0	1	-1	-1	-1	-1	-1	-1	1	-1	-1	1
PCHIC	SLATEC	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	0	0	0	-1	0	-1
PCHIM	SLATEC	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1	1	-1
PCHSP	SLATEC	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1	-1	-1

We admit more than one entry per module in the feature table for a given class. This is necessary in cases when certain features interact. For example, in E1a module PCHEZ performs either interpolation by cubic splines with not-a-knot end conditions or by monotonicity-preserving C^1 cubics with end conditions obtained by estimating derivatives which are then adjusted to preserve local monotonicity. In this case, two distinct entries for PCHEZ are stored.

We have prepared extended problem/software data for a number of other GAMS problem classes, including C10a1 (Bessel functions J, Y, and H of real argument and integer order), D2a1 (Solution of systems of linear equations, including matrix inverses, and LU and related decompositions for real nonsymmetric matrices), I2b1a (Second order linear elliptic boundary value problems, and I2b1a1a (Poisson or Helmholtz equation on a rectangular domain).

Finally, additional textual data describing each feature and feature type could also be stored in the knowledge base to allow advisors to provide users with further help in feature selection.

3 Selection Heuristics

We next describe a set of heuristics which can be used with the extended problem/software knowledge to aid in the selection of appropriate mathematical software. We assume that we are positioned at a leaf node of the problem taxonomy at which many modules have been classified. Our goal is to present the user with as few modules as possible. Initially, all modules are candidates. We then ask the user a sequence of questions, each of which has the potential of reducing the number of modules. The strategy is roughly as follows:

```
Module Reduction Strategy
mlist ← list of all modules in class;
foreach feature type until (length(mlist)==1 or user is satisfied) do:
    flist ← list of features of this type possessed by modules in mlist;
    prompt user for desired features from flist;
    eliminate inappropriate modules from mlist;
    if (length(mlist)==0) recover;
```

Note that only those features possessed by modules in the current module list are displayed at each stage. In this way the user need only make decisions about features which make a difference.

The advisor maintains a requirements vector which encodes the user's desires. The *i*th element of this vector gives the user's opinion of the *i*th feature. The possible values are:

1	feature is required
0	no opinion
-1	feature is forbidden

Initially, the requirements vector is zeroed. At each stage of module reduction, entries corresponding to the current feature type may be changed. In Table 6 we present a truth table which may be used to evaluate the user's opinion of a particular feature. A rejection of any feature is ground for rejecting the module. Note that we cannot immediately reject a module whose optional feature is forbidden by the user. In this case there may be acceptable alternatives. Here we reject only when *all* optional features of a given type are forbidden by the user. This is based on the assumption that all alternatives are enumerated in the features of a given type.

It is possible that a user's requirements may exclude all remaining modules. This may mean that there is indeed no software which satisfies the user's requirements. However, it is often the

Table 6: Truth Table for Evaluating User Opinion of Feature

		User		
		1	0	-1
	1	Accept	Accept	Reject
Module	0	Accept	Accept	Conditional
	-1	Reject	Accept	Accept

case that requirements can be relaxed by the user, and we have a recovery strategy to handle this. When all modules fail we present the user with a list of the current requirements, indicating in each case the number of modules which become available if the requirement is relaxed. The user may relax requirements until modules become available, in which case the reduction strategy resumes at the smallest numbered feature type which was relaxed. The availability of such a backtracking scheme makes the order in which questions are asked of the user less important.

More than one module may remain applicable, even after all user requirements have been ascertained. In this case, information implied by the requirements vector can be used to order the remaining modules so that the most likely matches appear first. For each module we count the number of features required by the module (extra properties) and the number of optional module features about which the user has no opinion (extra options). We then sort the modules in increasing order of extra properties; ties are sorted in increasing order of extra options. In this way the modules presented first are the closest match to user requirements.

4 Example

In this section we present a sample user session for a prototype advisory system which utilizes the knowledge base for class E1a and the selection heuristics outlined in the previous section. User responses are preceded by >>>.

```
There are   35 possible module selections.  Would you like to try to reduce this? >>> y

Which type of problem?

    1 : piecewise polynomial (PP) interpolation             [ 29 modules]
    2 : osculatory PP interpolation                         [  5 modules]
    3 : optimal interpolation (determine range of values)   [  1 modules]

Enter number(s) of required items, -number for items that must not be present.
If you have no opinion, then simply press return. >>> 1

There are now  29 possible module selections.  Each satisfies the following:

    1. type of problem is piecewise polynomial (PP) interpolation

Would you like to try to further reduce the number of module selections? >>> y

Which visual property?
```

```
     8 : as smooth as possible (spline)                    [ 16 modules]
     9 : preserve convexity                                [  1 modules]
    10 : preserve monotonicity                             [  5 modules]
    11 : Akima's smoothing method                          [  2 modules]

Enter number(s) of required items, -number for items that must not be present.
If you have no opinion, then simply press return. >>> 8 10 -11

No modules satisfy all your requirements, which are :

     1. type of problem is piecewise polynomial (PP) interpolation [  2 modules]
     8. visual property is as smooth as possible (spline)          [  5 modules]
    10. visual property is preserve monotonicity                   [ 16 modules]
    11. visual property is NOT Akima's smoothing method            [  0 modules]

The number in brackets tells you how many selections become available if this
requirement is removed.  Which, if any, of these can be removed? >>> 8 11

There are now  5 possible module selections.  Each satisfies the following:

     1. type of problem is piecewise polynomial (PP) interpolation
    10. visual property is preserve monotonicity

Would you like to try to further reduce the number of module selections? >>> y

All remaining module selections have the following polynomial degree : cubic
Is this acceptable? >>> y

Which end condition?

    13 : user-supplies derivatives                         [  1 modules]
    14 : software estimates derivatives                    [  5 modules]
    15 : not-a-knot (extra continuity near ends)           [  1 modules]
    17 : compatible with monotonicity                      [  5 modules]

Enter number(s) of required items, -number for items that must not be present.
If you have no opinion, then simply press return. >>> 14 17

There are   5 modules to display.  Each satisfies the following:

     1. type of problem is piecewise polynomial (PP) interpolation
     5. polynomial degree is cubic
    10. visual property is preserve monotonicity
    14. end condition is software estimates derivatives
    17. end condition is compatible with monotonicity

Modules which most closely match your requirements will be listed first.

E01BEF in NAG                   (Module   1 of   5)

PCHEZ in NMS                    (Module   2 of   5)
```

```
E1AM in the Scientific Desk      (Module  3 of  5)

PCHIM in SLATEC                  (Module  4 of  5)

PCHIC in SLATEC                  (Module  5 of  5)

   Additional properties :
     13. end condition is user-supplies derivatives (optional)
     15. end condition is not-a-knot (extra continuity near ends) (optional)
```

In this example, the original 35 possibilities are pared down to 5. If the user had instead selected features 13 or 15 in the last query then all but one would have been eliminated. The five modules remaining are, in fact, very similar. All but PCHIC are versions of Fred Fritsch's routine PCHIM. PCHIC, also developed by Fritsch, is an extension of PCHIM with additional control over end conditions. At this point other selection criteria already present in our system, such as which machine the software is available on, its portability, and precision could now be applied. For example, at NIST the E1AM and PCHEZ are available only on PCs, E01BEF is available only on our central mainframes, while PCHIM and PCHIC are available on local mainframes, minicomputers, workstations, and can be easily ported to PCs.

5 Conclusions

We have presented knowledge representation techniques and heuristics which can be used to enhance the performance of mathematical software selection systems based upon problem taxonomies. The extended problem/software data provides a detailed representation of the distinguishing features of software for a particular problem. The representation is quite compact, although an extensive survey of existing software is required to generate it. This is not surprising; if the system is to perform at the level of a human expert, then it is not unreasonable to require a human to teach it. Fortunately, the required information is local to a given problem, and hence need only be developed for problem classes in which an overabundance of software exists.

The selection heuristics we have developed allow an advisory system to efficiently find the closest match between user requirements and software features. Our heuristics do address the case in which the user inadvertently eliminates all software modules from consideration. Finally, our heuristics allow a selection system to easily explain why any given module is or is not acceptable. We have developed a prototype software advisor based on these heuristics which we plan to integrate into the GAMS Interactive Consultant in the near future. We believe that these techniques are sufficient to allow a software advisor to perform at the level of a mathematical software generalist.

Several areas have not yet been adequately addressed. We have not attempted to provide support for deep mathematical knowledge required to support selection based upon an evaluation of the software's performance. This is a much more difficult problem for which there have been some recent advances [10]. We have also not addressed the problem of what to do when there are really no suitable software modules. In this case we must find a way to help users reformulate their problem in such as way as to allow software for some other problem to be used, if possible.

Acknowledgements

I am grateful for helpful discussions with Jeanne Springmann and Sally Howe of NIST.

Disclaimer

Reference to commercial products in this paper does not constitute recommendation or endorsement by NIST.

References

[1] R. F. Boisvert. The Guide to Available Mathematical Software advisory system. *Math. & Comp. in Simul.*, 31:453–464, 1989.

[2] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. GAMS—a framework for the management of scientific software. *ACM Trans. Math. Softw.*, 11:313–355, 1985.

[3] R. F. Boisvert, S. E. Howe, and D. K. Kahaner. The Guide to Available Mathematical Software problem classification system. NISTIR 4475, National Institute of Standards and Technology, Gaithersburg, MD 20899, December 1990.

[4] R. F. Boisvert, S. E. Howe, and J. L. Springmann. Internal structure of the Guide to Available Mathematical Software. NISTIR 89-4042, National Institute of Standards and Technology, Gaithersburg, MD 20899, March 1989.

[5] J. Chelsom, D. Cornali, and I. Reid. Numerical and statistical knowledge-based front-ends research and development at NAG. In J. R. Rice, E. N. Houstis, and R. Vichnevetsky, editors, *Intelligent Mathematical Software Systems II.* IMACS, 1991. to appear.

[6] J. Corones. SLADOC — "expert" assistance for users of SLATEC routines. *SIAM News*, 21(3):20–21, May 1988.

[7] P. W. Gaffney, C. A. Addison, B. Andersen, S. Bjørnestad, R. E. England, P. M. Hanson, R. Pickering, and M. G. Thomason. NEXUS: Towards a problem solving environment (PSE) for scientific computing. *SIGNUM Newsletter*, 21(3):13–24, 1986.

[8] P. W. Gaffney, J. W. Wooten, K. A. Kessel, and W. R. McKinney. NITPACK: An interactive tree package. *ACM Trans. Math. Softw.*, 9:395–417, 1983.

[9] P. Hazel and M. R. O'Donohoe. Help Numerical: The Cambridge interactive documentation system for numerical methods. In M. A. Hennell and L. M. Delves, editors, *Production and Assessment of Numerical Software*, pages 367-382. Academic Press, London, 1980.

[10] M. Lucks and I. Gladwell. Automated selection of mathematical software. Computer Science Report 90–CS–26, Southern Methodist University, Dallas, TX 75275, 1990.

[11] J. C. Mason and I. Reid. Numerical problem-solving environments — current and future trends. In J. C. Mason and M. G. Cox, editors, *Scientific Software Systems*, pages 223–237. Chapman and Hall, London, 1988.

[12] K. Schulze and C. W. Cryer. NAXPERT: A prototype expert system for numerical software. *SIAM J. Sci. Stat. Comp.*, 9:503–515, 1988.

Fractals in Quaternions and their Application to Computer Graphics

Masaaki Shimasaki*, Yasuhiro Matsusaka** and Masanobu Nomura**
Computer Center, Kyushu University*
Department of Computer Science and Communication Engineering
Kyushu University**

Abstract

We discuss fractals in quaternions and their application to computer graphics. First we describe computation and visualization of Mandelbrot set for $f(z) = z^3 + q$ in quaternions.

Quaternions can be conveniently used to represent operations in three dimensional space, including rotations, projections and affine transformations. It is shown that a three dimensional tree-like pattern can be generated using similar contraction mapping in terms of quaternion functions to a triangle in the three dimensional space. Our quaternion functions consist of components for contraction, rotation and reflection in the three dimensional space. Ray-tracing technique is used to generate a scene with trees.

1 Introduction

With a simple mathematical formula and process, fractals, surprisingly mysterious and sophisticated graphic patterns, can be generated and much attention has been paid to fractals not only in mathematics but also in computer graphics. Fractals in three dimensional space or higher dimensional space are theoretically possible, but so far mainly fractals in two dimensional space have been investigated in complex dynamics. In computer graphics, it is quite natural to seek fractals in three dimensional space but as a rather rare and exceptional case in three dimensional space, fractal techniques has been applied to create mountain views and cliff views[1][2]. In this paper we consider fractals in higher dimensional space related to quaternions and their application to computer graphics. Quaternions have long been known, but it is quite recent that its importance in robotics and computer graphics was recognized and extensive research has been carried out in these two or three years. There are already papers on fractal patterns produced by quadratic quaternions functions set[2] and the Julia set for the quaternion iteration function $z^2 + q$ set[3]. We discuss the Mandelbrot set for $z^3 + q$ in quaternions. Computation of fractals is simple but cpu-time consuming. Using a super computer FACOM VP-2600/10, the cpu-time could be reduced to about one fifteenth of that in scalar mode. Visualization of fractals in higher dimensional space is an interesting example of visualization in scientific computation.

Quaternions can be conveniently used to represent operations in three dimensional space, including rotations, projections and affine transformations. As von Koch's two dimensional self-similar curve or Hata[4]'s cedar leave pattern in a plane is generated by repeated application of two similar contraction mappings to a triangle, we will show in the section 3 that a three dimensional tree-like pattern can be generated using similar contraction mapping to a triangle in the three dimensional space. While the mapping functions for von Koch's curve are complex functions, our similar mapping functions are quaternion functions. Our quaternion functions consist of components for contraction, rotation and reflection in the three dimensional space. Ray-tracing technique is used to generate a scene with trees.

2 Mandelbrot Set in Quaternions

2.1 Quaternions and the Mandelbrot Set

A quaternion value q is a four-tuple consisting of one real part and three imaginaries,

$$q = q_r + q_i i + q_j j + q_k k \tag{1}$$

where i, j and k are imaginary units and the following equations hold:

$$\begin{aligned} i^2 &= j^2 = k^2 = -1, & (2)\\ jk &= i, \quad ki = j, \quad ij = k, & (3)\\ kj &= -i, \quad ik = -j, \quad ji = -k. & (4) \end{aligned}$$

Mathematically quaternions form a noncommutative division ring. We define the norm of q by

$$||q|| = \sqrt{q_r^2 + q_i^2 + q_j^2 + q_k^2} \tag{5}$$

It should be noted that

$$q^2 = q_r^2 - q_i^2 - q_j^2 - q_k^2 + 2q_r(q_i i + q_j j + q_k k) \tag{6}$$

and, if $q_r = 0$ then, we have

$$q^2 = -q_i^2 - q_j^2 - q_k^2. \tag{7}$$

As a natural extension of the Mandelbrot set in the complex plane, we treat the following Mandelbrot set M in quaternions:

$$M = \{\mu : \lim_{n\to\infty} f_\mu^n(z_n) \not\longrightarrow \infty, \quad f'_\mu(q_c) = 0\} \tag{8}$$

where

$$f_\mu(q) = q^3 + \mu \tag{9}$$

and $f_\mu^n(q)$ denotes $f_\mu(f_\mu(...(f_\mu(q))...))$. In this particular case of the function $f_\mu(q)$, we have

$$q_c = 0. \tag{10}$$

In actual computation, we define the sequence $\{q_n\}$ by

$$\begin{aligned} q_0 &= 0 & (11)\\ q_{n+1} &= q_n^3 + \mu, \qquad n = 0, 1, 2, \cdots, n_{max}. & (12) \end{aligned}$$

If the following condition is satisfied for some $n \leq n_{max}$,

$$||q_{n+1}|| > 1 + ||\mu|| \tag{13}$$

then we consider the sequence $\{q_n\}$ has diverged. For a particular value of μ, we can compute the value of n for which the divergence condition is satisfied first, or $n = n_{max}$,where the sequence does not diverge. If $n = n_{max}$, we consider $n = 0$. If we define the four dimensional mesh points with equidistance, satisfying:

$$\begin{aligned}
\mu_{rmin} &\leq \mu_r \leq \mu_{rmax} && (14)\\
\mu_{imin} &\leq \mu_i \leq \mu_{imax} && (15)\\
\mu_{jmin} &\leq \mu_j \leq \mu_{jmax} && (16)\\
\mu_{kmin} &\leq \mu_k \leq \mu_{kmax} && (17)
\end{aligned}$$

then we can compute the value of the above n for each mesh point. Thus we can obtain the four dimensional integer array of n, where each element satisfies $0 \leq n \leq n_{max}$. We define color for each point according to the value of $IQ = mod(n, 7)$ as shown in Table1. We obtain the color chart of four dimensional space. It is not easy to visualize the four dimensional space, we give a series of two dimensional cross section of the four dimensional space shown in Figs.1 $\sim$ 5 .

Table 1: Color mapping table for IQ

IQ	0	1	2	3	4	5	6
color	black	blue	green	cyan	red	magenta	yellow

2.2 Computation of the Mandelbrot Set on Supercomputer

Although computation of the Mandelbrot set is conceptually very simple, it is very computationally intensive. Computation for individual mesh points are logically independent and parallel computation can be carried out. On arithmetic pipeline vector computers, computations for multiple mesh points can be vectorized. For certain values of μ, the sequence $\{q_n\}$ diverges for a small value of n. Once computation for a certain point diverges, then the point is removed from the set of points for computation. Namely the vector length of computation is reducing as iteration proceeds. If we treat as many points as possible, the vector length becomes longer but the required amount of main memory becomes also larger. When we treat the four dimensional mesh points, then the required amount of memory is very large. Computation for mesh points on a two dimensional cross section plane is reasonable both for vector length and for required amount of main memory. Table 3 gives an example of CPU time on FACOM VP-2600/10 for computation of the Mandelbrot set given in Fig.1. The ratio of CPU time of the vector mode to the scalar mode attains 15 times and it shows the effectiveness of vector computation.

Table 2: CPU time for computation of the Mandelbrot set in Fig. 1

number of mesh point	CPU time for scalar mode: T_s	CPU time for vector mode: T_v	ratio of CPU time $\frac{T_s}{T_v}$
1024 x 1024	37.8[sec]	2.40[sec]	15.8

2.3 Perspective View of the Three Dimensional Cross Section

A three-dimensional cross section of the Mandelbrot set in the four dimensional space is also given in Fig.6, which is produced by the ray-tracing technique.

3 Generation of Fractals in Quaternion and its Application to Computer Graphics

3.1 Vectors in Three Dimensional Space as Quaternions

Suppose that

$$q_r = 0, \tag{18}$$

$$q_i = x, \quad q_j = y, \quad q_k = z, \tag{19}$$

then the quaternion q

$$q = ix + jy + kz \tag{20}$$

represents a point (x, y, z) in the three dimensional space. If a quaternion q represents a unit vector, then we have $q^2 = -1$.

The largest merit of quaternions is their ability to represent rotations in the three dimensional space. Let us denote the axis of rotation by n, where $n_r = 0$. We assume that rotation of the vector v, where $v_r = 0$, around the axis n by the angle θ gives the vector u, where $u_r = 0$. It is known that the following equation holds:

$$u = qvq^{-1} \tag{21}$$

where

$$q = \cos\frac{\theta}{2} + in_x \sin\frac{\theta}{2} + jn_y \sin\frac{\theta}{2} + kn_z \sin\frac{\theta}{2}, \tag{22}$$

or in short

$$q = \cos\frac{\theta}{2} + n\sin\frac{\theta}{2} \tag{23}$$

From the basic equations on i, j and k, it can be easily verified that

$$q^{-1} = \cos\frac{\theta}{2} - in_x \sin\frac{\theta}{2} - jn_y \sin\frac{\theta}{2} - kn_z \sin\frac{\theta}{2}, \tag{24}$$

or

$$q^{-1} = \cos\frac{\theta}{2} - n\sin\frac{\theta}{2} \tag{25}$$

It should be noted that the representation of rotation by quaternions is a very concise form and there is no redundancy. A rotation in the three dimensional space can be represented by a 3 × 3 matrix. It requires 9 elements and redundancy of the representation is much larger than quaternion representation.

Projection and reflection of a vector can be also conveniently represented using quaternions. If v is a vector(i.e. $v_r = 0$), and n is a unit vector, then we have the following:

1. The projection of v onto Line(n) is given by $\frac{v-nvn}{2}$.
2. The projection of v onto Plane(n) is given by $\frac{v+nvn}{2}$.
3. The reflection of v across Line(n) is given by $-nvn$.
4. The reflection of v across Plane(n) is nvn.

The situation is illustrated in Fig.7. It should be noted that representation of projection and reflection is again very compact.

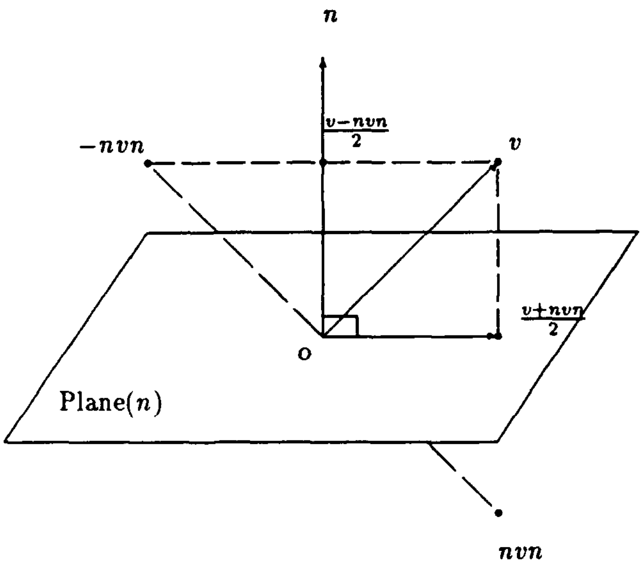


Fig. 7: Projection and reflection

3.2 Generation of Three Dimensional Fractals Using Quaternions and its Application to Computer Graphics

Koch's curve is a famous example of fractals in the two dimensional complex plane, which can be generated by repeated application of two self-similar contraction mapping functions to a triangle. Hata showed that a cedar leave-like pattern can be generated by two complex functions similar to Koch's function to a triangle. Hata's functions are given by:

$$F_1(z) = (\frac{1}{2} + \frac{\sqrt{3}}{6})\bar{z} \qquad (26)$$

$$F_2(z) = \frac{2}{3}\bar{z} + \frac{1}{3} \qquad (27)$$

We now describe generation of fractal pattern in the three dimensional space. Let us consider $\triangle$ABC in Fig.8, where α and β are parameters and $0 < \alpha < 1$, $0 < \beta < 1$, $\alpha^2 + \beta^2 < 1$. We assume the following mapping process:

$\triangle ABC \rightarrow \triangle A'B'C$: rotation around the y-axis by the angle $\frac{\pi}{2} + \theta$.

$\triangle A'B'C \rightarrow \triangle A''B''C$: contraction with the scaling factor $\sqrt{\alpha^2 + \beta^2}$.

$\triangle A''B''C \rightarrow \triangle A'''B'''C$: reflection across the xy-plane.

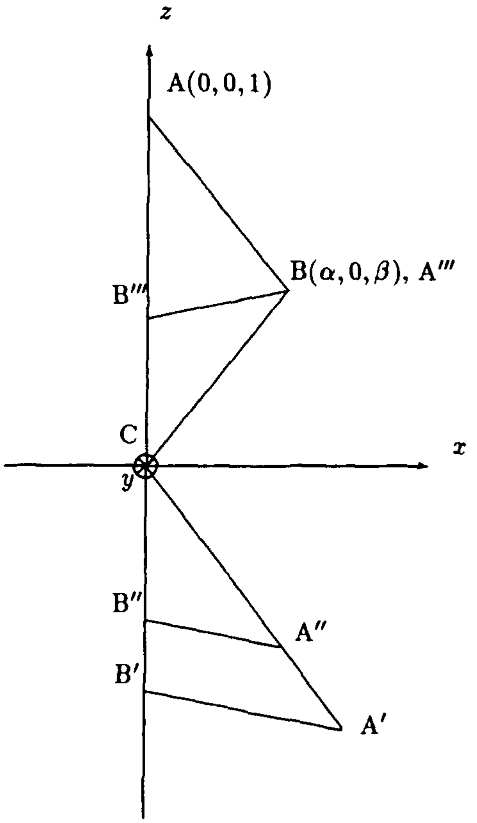


Fig. 8: Mapping by $f_1(v)$

If we denote the mapping function for the above process by $f_1(v)$ then, we have

$$f_1(v) = \sqrt{\alpha^2 + \beta^2} k q_1 v q_1^{-1} k \tag{28}$$

where

$$v = ix + jy + kz \tag{29}$$

$$q_1 = \cos(\frac{\pi}{4} + \frac{\theta}{2}) + j \sin(\frac{\pi}{4} + \frac{\theta}{2}) \tag{30}$$

$$\cos\theta = \frac{\alpha}{\sqrt{\alpha^2 + \beta^2}} \tag{31}$$

$$\sin\theta = \frac{\beta}{\sqrt{\alpha^2 + \beta^2}} \tag{32}$$

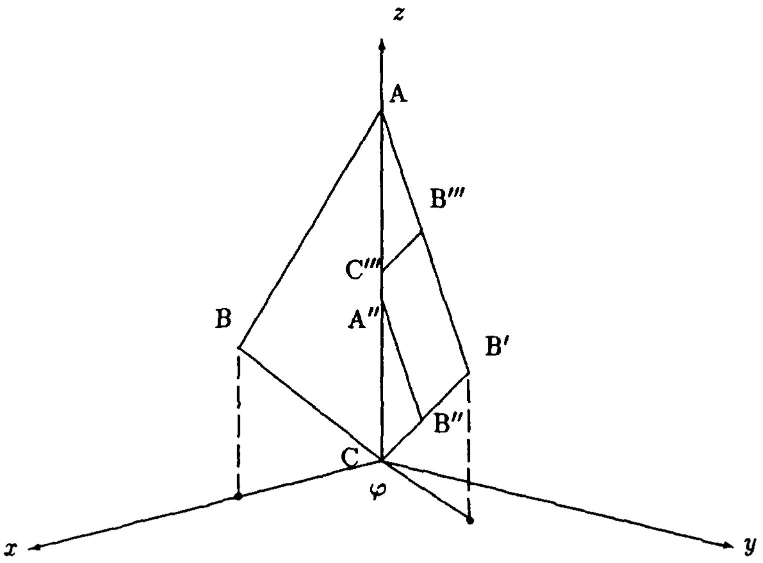


Fig. 9:Mapping by $f_2(v)$

Let us consider the following mapping process shown in Fig.9.

$\triangle ABC \rightarrow \triangle AB'C$: rotation around the a-axis by the angle φ.

$\triangle AB'C \rightarrow \triangle A''B''C$: contraction with the scaling factor $1-\alpha^2-\beta^2$.

$\triangle A''B''C \rightarrow \triangle AB'''C'''$: translation with the distance $\alpha^2+\beta^2$.

If we denote the mapping function for the above process by $f_2(v)$, then we have

$$f_2(v) = (1-\alpha^2-\beta^2)q_2 v q_2^{-1} + k(\alpha^2+\beta^2) \tag{33}$$

where

$$v = ix + jy + kz \tag{34}$$

$$q_2 = \cos\frac{\varphi}{2} + k\sin\frac{\varphi}{2} \tag{35}$$

We give Fig.10 as an example of three dimensional tree-like fractal patterns obtained by repeatedly applying $f_1(v)$ and $f_2(v)$ to $\triangle$ABC.

It should be noted that the fractal pattern in Fig.10 is a truly three dimensional fractal pattern and it is not the pattern which is obtained by rotation of a fractal pattern in the two dimensional space. The pattern in Fig.10 is an interesting pattern in itself. Using the ray-tracing technique, Fig.11 was produced using the pattern in Fig.10.

4 Conclusion

We give a series of two dimensional cross section views of the four dimensional Mandelbrot set for $f(z) = z^3 + q$, and a perspective view of the three dimensional cross section. Although computation of the Mandelbrot set is cpu-time consuming, the computation can be carried out efficiently on supercomputers.

It was shown that a three dimensional tree-like fractal pattern can be generated using similar contraction mappings in terms of quaternion functions. The ability of quaternions to represent operations in the three dimensional space was effectively used. Further applications of quaternions to computer graphics seems to be promising.

References

[1] Alain Fournier, Don Fussell, Loren Carpenter : Computer Rendering of Stochastic Models, Comm. ACM Vol. 25, June 1982, pp.371-384.

[2] Benoit B. Mandelbrot : The fractal geometry of nature. W. H. Freeman and Company, New York, 1983.

[3] John C. Hart, Daniel J. Sandin, Louis H. Kauffman : Ray Tracing Deterministic 3-D Fractals. Computer Graphics, Vol. 23, Num. 3, July 1989, pp.289-296.

[4] M. Hata : On the Structure of the Self-Similar Sets. Japan J, Appl, Math. 2, 1985, pp.381-414.

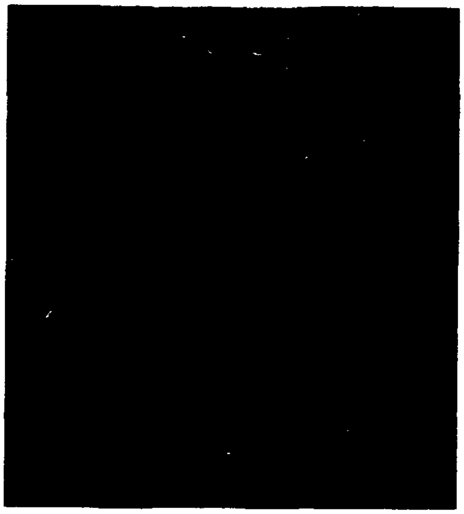


Fig. 1: $-1.35 \le \mu_r \le 1.35, \quad -1.35 \le \mu_i \le 1.35,$

$\mu_j = 0.6, \quad \mu_k = 0$

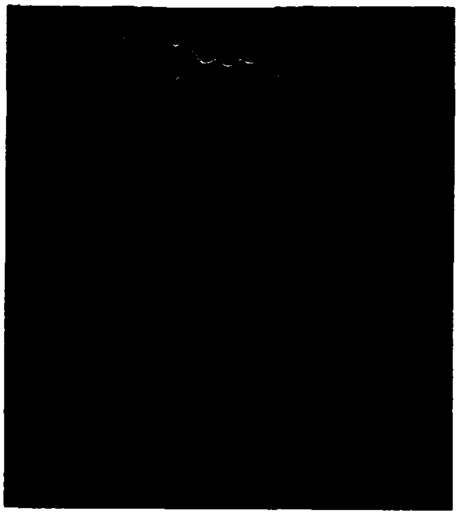


Fig. 2: $-1.35 \le \mu_r \le 1.35, \quad -1.35 \le \mu_i \le 1.35,$

$\mu_j = 0.7, \quad \mu_k = 0$

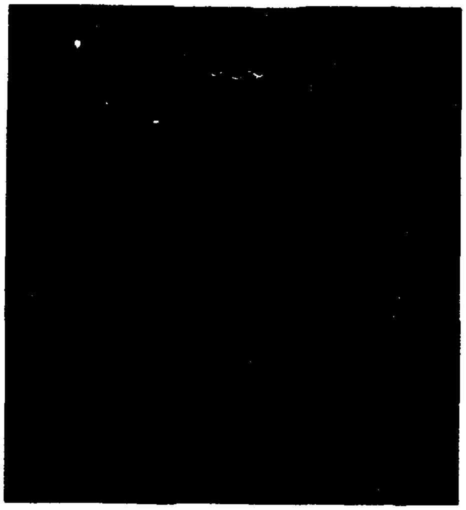


Fig. 3: $-1.35 \le \mu_r \le 1.35, \quad -1.35 \le \mu_i \le 1.35,$

$\mu_j = 0.8, \quad \mu_k = 0$

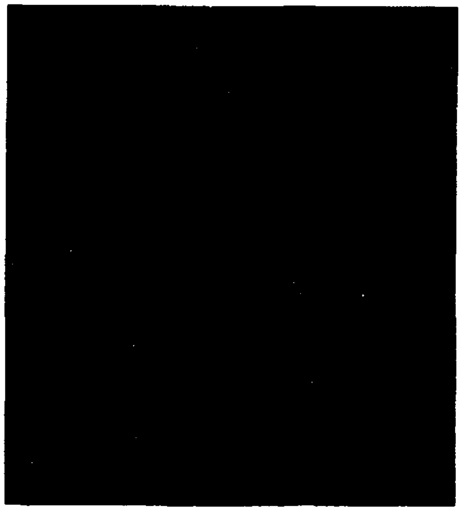


Fig. 4: $-1.35 \le \mu_r \le 1.35, \quad -1.35 \le \mu_i \le 1.35,$

$\mu_j = 1.1, \quad \mu_k = 0$

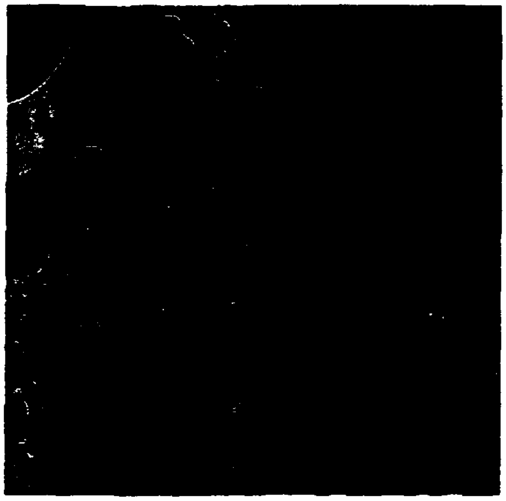


Fig. 5: $-0.250 \leq \mu_r \leq -0.225, \quad 0.590 \leq \mu_i \leq 0.615,$

$\mu_j = 0.7, \quad \mu_k = 0$

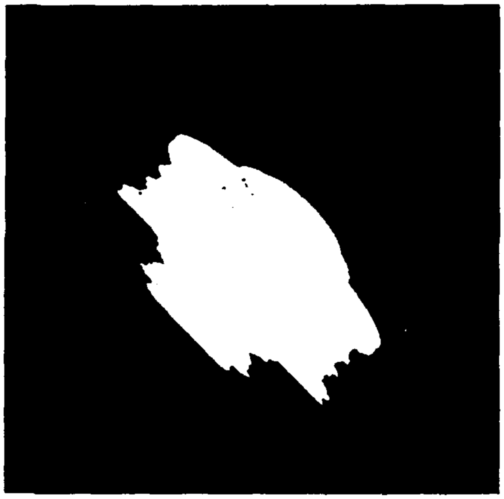


Fig. 6: $-1.35 \leq \mu_r \leq 1.35, \quad -1.35 \leq \mu_i \leq 1.35,$

$0.0 \leq \mu_j \leq 1.35, \quad \mu_k = 0$

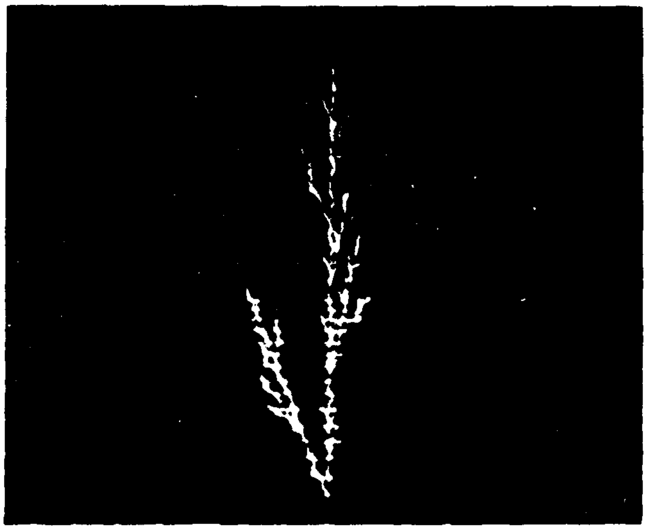


Fig. 10: $\alpha = 0.2, \quad \beta = 0.5$

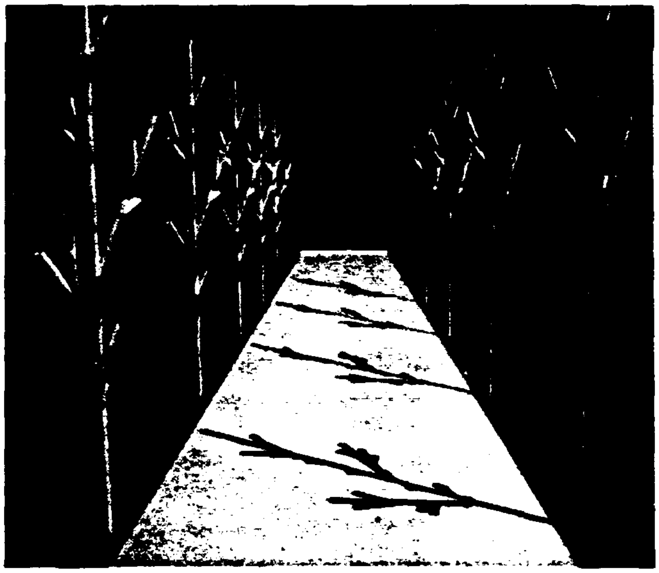


Fig. 11: $\alpha = 0.2, \quad \beta = 0.5$

Computer Algebra Tools for Higher Symmetry Analysis of Nonlinear Evolution Equations

V.P.Gerdt
Joint Institute for Nuclear Research
Head Post Office P.O.Box 79, Moscow, USSR

Abstract

This paper presents a computer-aided approach and a software package for symbolic algebraic computation to solve the problem of verifying the existence of the canonical Lie-Bäcklund symmetries for multicomponent quasilinear evolution equations with polynomial-nonlinearity and computing of a given order symmetry if it exists. In the presence of arbitrary numerical parameters the problem is reduced to investigation and solving of nonlinear algebraic equations in those parameters. It is remarkable that in all the known cases these algebraic equations are completely solvable by the Gröbner basis technique implemented as a part of the software package.

1 Introduction

The symmetry analysis of differential equations is one of the central problems in modern applied mathematics and mathematical physics. Among numerous methods of analysis and integration of differential equations the most general and universal ones are based on their symmetry properties. S.Lie has introduced the concept of the symmetry just for the purpose of creating solutions of differential equations. From the theoretical point of view the problems of symmetry analysis are investigated in sufficient detail. But in practice to find the symmetry group (or even some individual generators) of a given differential equation it is necessary to carry out extremely tedious algebraic manipulations. That is why computer algebra has continued to play an increasingly important part in the practical symmetry analysis [1].

Now there are several computer algebra packages for symmetry analysis of differential equations. Among them the big packages SODE for ordinary differential equations and SPDE for partial differential equations are the best developed [1]-[2] for determining so-called *classical* or *point* or *Lie* symmetries. They use the most general method of computation which based on generating and solving of the determining system in the form of linear differential equations in functions which occur in the definition of a symmetry generator. Both Reduce and Scratchpad II versions of the packages SODE and SPDE have been designed according to basic concepts of software engineering. Moreover, data abstraction as one of the main attributes of the Scratchpad II system allowed one to gain very effective module organization of the package with the detailed investigation of its complexity [2]. The most difficult part of the whole computational process is simplification and integration of the determining equations. At this step a user has often to do a reasonable ansatz on the structure of symmetries. By this reason an interactive regime is always assumed.

In the searching of so-called *generalized* or *higher* (*Lie-Bäcklund*) symmetries, when functions which occur in the definition of a symmetry generator may depend not only on the point, i.e., the dependent and the independent, variables but also on the derivatives of the unknown functions, on an appropriate ansatz plays still further important role. The point is that the existence of a higher symmetry imposes much more strong limitations on the equations under consideration than the existence of the classical Lie symmetries. By this reason a universal computer algebra package for the construction of higher-order symmetries based on the most general scheme of computation (see, for example [3]) m ay not be usable for many nonlinear problems. Therefore the special constructive and effective methods for finding the generalized symmetries in some sufficiently wide class of nonlinear differential equations are of interest for the design of the corresponding computer algebra packages.

In this paper a computer-aided approach to construction of higher symmetries is presented which can be applied to wide class of multicomponent quasilinear partial differential equations of the evolution type. After necessary mathematical definitions and formulae (Sect.2), description of the computational procedure for higher symmetry analysis (Sect.3) and its implementation (Sect.4) in the form of package written in internal language (Rlisp) of the Reduce computer algebra system are given. The package consists of the two functionally independent modules. One of them is destined for the symmetry analysis proper and another for solving systems of nonlinear algebraic equations which arise in the presence of arbitrary numerical parameters. As an illustration, the computation of the third order Lie-Bāclund symmetries for eight-parametric family of coupled nonlinear Schrödinger equations is considered (Sect.5).

2 Mathematical Background

Among the partial differential equations of physical interest, an important role is played by the class of polynomial-nonlinear evolution equations (NLEE) in one-spatial and one-temporal dimension of the following form

$$\begin{array}{l} u_t = \Phi(x,u,\ldots,u_N) = \Lambda u_N + F(x,u,\ldots,u_{N-1};\alpha_1,..\alpha_K),\ \ N \geq 2 \\ u = u(t,x), u_k = \partial^k u/\partial x^k, u = (u^1,\ldots,u^M), F = (F^1,\ldots,F^M), \\ \Lambda = diag(\lambda_1,\ldots,\lambda_M),\ \ \lambda_i,\alpha_i \in \mathbf{C},\ \ \lambda_i \neq 0, \end{array} \qquad (1)$$

where the vector function F is a polynomial in its arguments including numeric parameters α_i if they are. F is said to be a differential function of $N-1$ order. The function Φ has the order N respectively.

The class (1) contains such well-known *integrable* NLEE as the Korteweg-de Vries equation, the Burgers equation, the nonlinear Schrödinger equation and many other ones which are now under intensive investigation.

The concept of *integrability* is closely connected with the existence of the higher symmetries [4]: NLEE is integrable if and only if it possesses infinitely many time-independent higher symmetries. But in practice the existence of M different higher symmetries is sufficient for integrability of M-component NLEE.

Definition. A vector function $H = (H^1, \ldots, H^M)$ of a finite number of differential variables $x, u, u_1, \ldots, u_n$ is a n-order (higher) *symmetry* of the system (1) if it leaves (1) invariant under the transformation $t' = t$, $x' = x$, $u' = u + \tau H(x, u, u_1, \ldots, u_n)$ within order τ. This means that H corresponds to the canonical Lie-Bäcklund operator [5]

$$X = \sum_{i=1}^{M} H^i \frac{\partial}{\partial u^i} + \cdots \tag{2}$$

and satisfies the differential equation

$$\frac{dH}{dt} = \Phi_*(H),$$

which is equivalent to the operator relation

$$\frac{dH_*}{dt} - [H_*, \Phi_*] = \frac{d\Phi_*}{d\tau}. \tag{3}$$

Here Φ_* and H_* are matrix differential operators

$$\Phi_* = \sum_{i=0}^{N} \Phi_i D^i, \ \ [\Phi_i]_{kj} = \partial\Phi^k/\partial u_i^j, \ \ H_* = \sum_{i=0}^{n} H_i D^i, \ \ [H_i]_{kj} = \partial H^k/\partial u_i^j \tag{4}$$

and

$$D = \frac{d}{dx} = \frac{\partial}{\partial x} + \sum_{i=1}^{M}\sum_{j=0}^{\infty} u_{j+1}^i \frac{\partial}{\partial u_j^i}, \tag{5}$$

$$\frac{d}{dt} = \sum_{i=1}^{M}\sum_{j=0}^{\infty} D^j(\Phi^i)\frac{\partial}{\partial u_j^i}, \qquad \frac{d}{d\tau} = \sum_{i=1}^{M}\sum_{j=0}^{\infty} D^j(H^i)\frac{\partial}{\partial u_j^i}$$

are the total differentiation operators with respect to x, t and τ respectively.

3 Construction of Higher Symmetries

To compute higher-order ($n > N$) symmetries for a given NLEE of the form (1) the effective algorithms have been developed [6,7,8] which take into account the basic methods being used by experts [9] in their pencil and paper work.

The basic idea is to construct step by step the coefficients A_i, $i = n, n-1, \ldots, 0$ of matrix differential operator

$$L = A_0 + A_1 D + \cdots + A_n D^n \tag{6}$$

as a solution of the operator equation

$$\frac{dL_*}{dt} - [L_*, \Phi_*] = \frac{d\Phi_*}{d\tau} = \frac{d\Phi_{N-1}}{d\tau} D^{N-1} + \cdots. \tag{7}$$

which corresponds to relation (3) with constant diagonal matrix $\Phi_N = \Lambda$ as defined in (1).

Isolation of coefficients of D^i in the operator equality (7) gives the following chain of equations in A_i

$$\begin{array}{rc}
D^{N+n}: & [\Lambda, A_n] = 0, \\
D^{N+n-1}: & N \cdot \Lambda \cdot D(A_n) + [\Lambda, A_{n-1}] + [\Phi_{N-1}, A_n] = 0, \\
\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
D^{N+n-i}: & N \cdot \Lambda \cdot D(A_{n-i+1}) + [\Lambda, A_{n-i}] + [\Phi_{N-1}, A_{n-i+1}] + B_i = 0, \\
\ldots\ldots\ldots\ldots & \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\
D^N: & N \cdot \Lambda \cdot D(A_1) + [\Lambda, A_0] + [\Phi_{N-1}, A_1] + B_n = 0,
\end{array} \tag{8}$$

where B_i is expressed in terms of A_j, $j > n - i + 1$.

The structure of matrix Λ in (1) and the form of i-th equation of chain (8) make possible finding the diagonal parts of A_{n-i+1} and non-diagonal parts of A_{n-i}. For example, in the case of different eigenvalues λ_i, from the first two equations of (8) it follows that A_n is arbitrary diagonal number matrix $A_n = diag(\mu_1, \mu_2, \ldots, \mu_M)$, $\mu_i \in \mathbf{C}$. General recurrent formulae for A_i as solutions of (8) are given in [6,7,8]. Because of this, equations (8) allow one to compute sequentially matrices $A_n, A_{n-1}, \ldots, A_1$ and non-diagonal part of A_0.

To provide the existence of a *local* higher symmetry $H(x, u, u_1, \ldots, u_n)$, chain (8) must admit *local*, i.e. depending on a finite number of dynamic variables taken from an infinite set $x, u, u_1, \ldots$, solutions A_i as well. From Eqs.(8) it follows that to find the diagonal part of A_i it is necessary to solve an equation of the form

$$D(Q) = S, \tag{9}$$

where operator D defined by expression (4). For a given local S, Eq.(9) admits a local solution $Q = D^{-1}(S)$ only if S satisfies a number of restrictions [6]. The reverse operator D^{-1} is none other than integration operator with respect to x. Hence at each step of chain (8) a number of arbitrary constants is generated. These constants may be important for analysis of the next steps.

After construction of the n-th order operator (6) by means of Eqs.(8) one can compute the n-th order symmetry using operator relation

$$H_* - diag(H_0) = \tilde{L} \equiv L - diag(A_0), \tag{10}$$

which follows from Eqs.(3) and (7). Operating by both sides of (10) on $u_1 \equiv u_x$ we obtain

$$\tilde{D}(H) = \tilde{L}(u_1), \quad \tilde{L}_j = D - \partial/\partial x - u_1{}^j \cdot \partial/\partial u^j \tag{11}$$

Eq.(11) defines the components H^j of symmetry H within arbitrary functions $h^j(u^j)$

$$H^j = \tilde{D}^{-1}(\tilde{L}u_1)^j + h^j(u^j). \tag{12}$$

Algorithms of D and $\tilde{D}$ reversion are described in [6]. They allow to verify the conditions of solvability of Eqs.(9) and (11)

$$S \in Im(D), \quad (\tilde{L}u_1)^j \in Im(\tilde{D}_j).$$

The notation $\rho \in ImD$ means that $\rho = D\sigma$ where σ is some local function. It is just solvability of (9) in terms of the corresponding local functions of chain (8) leads to the existence of higher symmetries for Eq.(1).

Because a higher symmetry of some fixed order may not exist for a given NLEE of form (1), the best computational strategy is the following one.

Step 1. Verification of the necessary conditions for the existence of higher symmetries. Those necessary conditions follow from solvability of Eq.(7) in terms of series (6) and have the form of the local conservation laws [6]-[7]

$$\frac{d}{dt}R(i,j) \in ImD, \; i = 0, 1, \ldots \; , j = 1, 2, \ldots, M \; . \tag{13}$$

Densities $R(i,j)$ in (13) are computed in terms of the r.h.s. of (1) [6]-[7]. For example, $R(0,j) = \partial F^j / \partial u^j_{N-1}$.

In the presence of arbitrary parameters α_i in (1) the necessary conditions (13) for a higher-order symmetry are equivalent to some system of nonlinear algebraic equations in those parameters. As an illustration, let us consider two-component case $u = (v, w)$ and the following local expression $\rho = a * v_2 * w + b * v * w_2 + c * v_1 * w_1$. The condition $\rho = D\sigma$ is solvable in terms of local function σ if and only if $c = a + b$. In that case $\sigma = a * v * w_1 + b * v_1 * w$.

In what follows we have to verify whether the obtained algebraic system has a solution. It is remarkable that the Gröbner basis technique [10], being the well-known tool of computer algebra, gives the most elegant and effective method for solving that problem.

Step 2. Previous step gives very important information on the existence of a higher symmetry. Now it is possible to try to construct the explicit form of the latter for some fixed order using the above algorithm. At this step we may obtain new restrictions on the r.h.s. of (1) in the form of algebraic equations in its parameters.

Step 3. Solving of the resulting system of nonlinear algebraic equations obtained at steps 1,2. Here the Gröbner basis technique again provides a means to simplify the problem drastically. Moreover, in many cases, in particular, in problems of classification of integrable NLEE [12], it allows to find all (even infinitely many) solutions in an explicit algebraic form.

4 Implementation in Reduce

We have implemented the above computational scheme for polynomial-nonlinear evolution equations (1) in the Reduce computer algebra system [13]. Our package consists of the two functionally different modules written in the language Rlisp of the Reduce symbolic mode.

The first module HSYM, which abbreviates Higher Symmetry, provides the procedures for sequential verifying the necessary conditions (13) in the case when there are no arbitrary parameters in initial NLEE (1). If they are HSYM generates an equivalent system of nonlinear algebraic equations. The solvability of the latter guarantees the existence of the higher-order conservation laws (13). Their densities $R(i,j)$ are computed in explicit form. HSYM has also the special procedure realizing the method of Sect.3 for finding the explicit form of the higher symmetry of the order specified by a user.

The restriction imposed in HSYM that F is a polynomial in its arguments, being very important from the viewpoint of applications, has given the possibility to establish the efficient algorithms for the realization of all necessary algebraic manipulations. They are based on the built-in recursive representation for polynomials in "standard form" and effectively use the corresponding built-in procedures acting at "standard forms" and "standard quotients" of the Reduce internal data.

The second module ASYS, which abbreviates Algebraic System, provides verifying the consistency of the systems of algebraic equations which arise at step 1 of Sect.3 as necessary conditions for the existence of higher symmetries. For this purpose it is sufficient to compute [10] a Gröbner basis G for an ideal generated by a set of polynomials under consideration. The system is unsolvable if $1 \in G$. ASYS contains the procedures for a Gröbner basis computation realizing well-known Buchberger's algorithm [10].

Solving the systems of algebraic equations at step 3 of Sect.3 is accomplished in ASYS as follows. A lexicographic Gröbner basis is constructed. Then ASYS computes the dimension and independent sets of variables for the ideal according to the method described in [11]. If our algebraic equations have infinitely many solutions the ideal has a positive dimension and the variables of each independent set can be considered as free parameters. In this case the obtained Gröbner basis is recomputed for each set of parameters leaving the order of the other variables unchanged. As a result a set of Gröbner bases is obtained with a simple structure and with "separated" variables (G is "triangularized") [10]. In this way the problem of solving a (often very complicated) system of nonlinear algebraic equations is always reduced to solving an equation in one variable.

In the general case only this last stage of computation may not be done automatically by our package. But our experience shows that the solutions can often be found with the help of the Reduce polynomial factorization facilities [13]. In the case of integrable NLEE their higher symmetry analysis leads to algebraic equations which can certainly be solved in completely algebraic way by using ASYS [12].

5 Example

As an example of application of our package let us consider the following eighth-parametric system of two coupled nonlinear Schrödinger equations.

$$\left\{ \begin{array}{l} i(\Psi_1)_t = \alpha_1(\Psi_1)_{xx} + \beta_1 | \Psi_1 |^2 \Psi_1 + \gamma_1 | \Psi_2 |^2 \Psi_1 + \delta_1 \Psi_2^2 \Psi_1^*, \\ i(\Psi_2)_t = \alpha_2(\Psi_2)_{xx} + \beta_2 | \Psi_2 |^2 \Psi_2 + \gamma_2 | \Psi_1 |^2 \Psi_2 + \delta_2 \Psi_1^2 \Psi_2^*. \end{array} \right. \tag{14}$$

Here Ψ_i are complex functions and α_i, β_i, γ_i, δ_i $(i = 1,2)$ are real parameters. This family of nonlinear evolution equations includes, for example, the systems describing interaction of electromagnetic waves with different polarizations in nonlinear optics [14] and resonant interaction of the long acoustic and short waves [15]. The complete integrability of (14) at $\gamma_1 = \gamma_2$ and $\delta_1 = \delta_2$ have been studied by another method in [16].

In order to be integrable (14) must have the higher symmetries of the order $n \geq 3$ of the form.

$$H_i(\Psi_j, (\Psi_j)_x, ..., (\Psi_j)_{x..x(n-times)}), \; i,j = 1,2; \; n \geq 3$$

which correspond to the canonical Lie-Băcklund operators (2).

Introducing the notations $u = \Psi_1$, $v = \Psi_1^*$, $p = \Psi_2$, $q = \Psi_2^*$, $\tau = it$ we can rewrite (14) in the form (1)

$$\begin{aligned} u_\tau &= \alpha_1 u_{xx} + \beta_1 u^2 v + \gamma_1 upq + \delta_1 vp^2 \ , \\ v_\tau &= -\alpha_1 v_{xx} - \beta_1 uv^2 - \gamma_1 vpq - \delta_1 uq^2 \ , \\ p_\tau &= \alpha_2 p_{xx} + \beta_2 p^2 q + \gamma_2 uvp + \delta_2 u^2 q \ , \\ q_\tau &= -\alpha_2 q_{xx} - \beta_2 pq^2 - \gamma_2 uvq - \delta_2 v^2 p \ . \end{aligned}$$

As a result of the first two necessary conditions, the module HSYM generates the three set of algebraic equations in dependence on the relation between α_1 and α_2 and under assumption that $\alpha_1\alpha_2 \neq 0$ in accordance with (1):

$$\begin{gathered} 1) \;\; \alpha_1 \neq \pm\alpha_2 \\ \alpha_1\beta_1\gamma_1 - \alpha_2\gamma_1\gamma_2/2 = \alpha_1\gamma_1\gamma_2 - 2\alpha_2\beta_2\gamma_2 = \alpha_1\gamma_1\delta_2 - \alpha_2\gamma_1\delta_2 = 0 \\ \alpha_1\gamma_2\delta_1 - \alpha_2\gamma_2\delta_1 = \beta_1\gamma_1\delta_2 - \gamma_2^2\delta_1/2 = \beta_1\beta_2\gamma_2 - \gamma_1\gamma_2^2/4 = 0 \\ \gamma_1^2\delta_2 - 2\beta_2\gamma_2\delta_1 = \gamma_1\gamma_2\delta_1 - 2\beta_2\gamma_2\delta_1 = \gamma_1\gamma_2\delta_2 - \gamma_2^2\delta_1 = 0 \\ \beta_1\delta_1 - \gamma_1\delta_2/2 = \beta_2\delta_2 - \gamma_2\delta_1/2 = 0 \end{gathered} \tag{15}$$

$$\begin{gathered} 2) \;\; \alpha_1 = \alpha_2 \\ \gamma_1\delta_1 - 2\beta_2\delta_1 = \gamma_1\delta_2 - \gamma_2\delta_1 = \beta_1\gamma_1 - \beta_2\gamma_2 = \beta_1\gamma_2 - \gamma_2^2 + 2\delta_2^2 = 0 \\ \beta_2^2\delta_1 - \delta_1^3 = \beta_2\gamma_2\delta_1 - 2\delta_1^2\delta_2 = \beta_2\delta_2 - \gamma_2\delta_1/2 = \gamma_2^2\delta_1 - 4\delta_1\delta_2^2 = 0 \\ \gamma_2^2\delta_2 - 4\delta_2^3 = \gamma_1^2 - \gamma_1\beta_2 - 2\delta_1^2 = \gamma_1\gamma_2 - \beta_2\gamma_2 - 2\delta_1\delta_2 = 0 \\ \beta_1\delta_1 - \gamma_2\delta_1/2 = \beta_1\delta_2 - \gamma_2\delta_2/2 = 0 \end{gathered} \tag{16}$$

$$\begin{gathered} 3) \;\; \alpha_1 = -\alpha_2 \\ \beta_1\gamma_1 - \beta_2\gamma_2 = \beta_1\gamma_2 + \gamma_2^2 = \beta_1\delta_1 = \gamma_1^2 + \gamma_1\beta_2 = \gamma_1\gamma_2 + \beta_2\gamma_2 = 0 \\ \gamma_1\delta_1 = \gamma_1\delta_2 = \beta_2\delta_2 = \gamma_2\delta_1 = \gamma_2\delta_2 = 0 \end{gathered} \tag{17}$$

Module ASYS allows readily to obtain all the solutions of (15)-(17). But construction of a symmetry according to algorithm of Sect.3 which are implemented in the module HSYM, may lead to new restrictions on the initial evolution equations in addition to those which follow from the necessary integrability conditions. In the case of polynomial-nonlinear evolution equations with arbitrary parameters HSYM allows one to produce an extra set of algebraic equations for a given order of a higher symmetry (see Sect.2,3). We omit here those extra equations because of their awkwardness.

Tables 1 gives all the solutions of (15)-(17) such that (6) possesses the Lie-Bäclund symmetries of the order $n \geq 3$. The corresponding third order symmetries are listed in Table 2.

Free variables	Solutions
1) $\alpha_1, \alpha_2, \beta_1, \beta_2$	$\gamma_1 = 0, \gamma_2 = 0, \delta_1 = 0, \delta_2 = 0.$
2) $\alpha_1, \beta_1, \beta_2$	$\alpha_2 = \pm\alpha_1, \gamma_1 = \pm\beta_2, \gamma_2 = \pm\beta_1, \delta_1 = 0, \delta_2 = 0.$
3) $\alpha_1, \delta_1, \delta_2$	$\alpha_2 = \alpha_1, \beta_1 = \pm\delta_2, \gamma_1 = \pm 2\delta_1, \beta_2 = \pm\delta_1, \gamma_2 = \pm 2\delta_2.$

Table 1: Subset of solutions of (15)-(17) which provides the existence of Lie-Bäcklund symmetries

Free variables	Symmetries
1) $\alpha_1, \alpha_2,$ β_1, β_2	$H_1 = \alpha_1(\Psi_1)_{xxx} + 3\beta_1(\Psi_1)_x \vert \Psi_1 \vert^2$ $H_2 = \alpha_2(\Psi_2)_{xxx} + 3\beta_2(\Psi_2)_x \vert \Psi_2 \vert^2$
2) $\alpha_1, \beta_1, \beta_2$	$H_1 = \alpha_1(\Psi_1)_{xxx} \pm 3/2\ \beta_2(\Psi_1\Psi_2)_x\Psi_2^* + 3\beta_1(\Psi_1)_x \vert \Psi_1 \vert^2$ $H_2 = \pm\alpha_1(\Psi_2)_{xxx} \pm 3/2\ \beta_1(\Psi_1\Psi_2)_x\Psi_1^* + 3\beta_2(\Psi_2)_x \vert \Psi_2 \vert^2$
3) $\alpha_1, \delta_1, \delta_2$	$H_1 = \alpha_1(\Psi_1)_{xxx} \pm 3(\Psi_1)_x(\delta_1 \vert \Psi_2 \vert^2 + \delta_2 \vert \Psi_1 \vert^2) + 3\delta_1(\Psi_2)_x(\Psi_2\Psi_1^* \pm \Psi_1\Psi_2^*)$ $H_2 = \alpha_1(\Psi_2)_{xxx} \pm 3(\Psi_2)_x(\delta_1 \vert \Psi_2 \vert^2 + \delta_2 \vert \Psi_1 \vert^2) + 3\delta_2(\Psi_1)_x(\Psi_1\Psi_2^* \pm \Psi_2\Psi_1^*)$

Table 2: Lie-Bäcklund symmetries of the third order for the solutions of Table 1

We conclude that all the systems of the form (1) possessing the canonical Lie-Bäcklund symmetries of the above structure are exhausted by Table 1. This conclusion is consistent with the results of Ref.[16]. The complete list of the third order symmetries is given in Table 2. Computation of the symmetries $1) - 3)$ with our Reduce package requires about 20, 40 and 50 seconds on an IBM PC AT-386 (25 Mhz) respectively. Other canonical Lie-Bäcklund symmetries of the order $n \geq 4$ can be found in a completely automatic way as well.

References

[1] Schwarz F. Symmetries of Differential Equations: from Sophus Lie to Computer Algebra, *SIAM Rev.* 30, 3, 450-481, 1988.

[2] Schwarz F. Programming with Abstract Data Types: The Symmetry Packages SODE and SPDE in Scratchpad, *Lecture Notes in Computer Science* 296, Springer-Verlag, Berlin, New-York, 1988, pp. 167-176.

[3] Fushchich W.I., Kornyak V.V. Computer Algebra Application for Determining Lie and Lie-Bäcklund Symmetries of Differential Equations, *J. Symb. Comp.* 7, 611-619, 1989.

[4] Fokas A.S. Symmetries and Integrability.*Stud. Appl. Math.*, 77, 253-299, 1987.

[5] Ibragimov N.H. Transformation Groups Applied to Mathematical Physics, Reidel, Boston, 1985.

[6] Gerdt V.P., Shabat A.B., Svinolupov S.I., Zharkov A.Yu. Computer Algebra Application for Investigating Integrability of Nonlinear Evolution Systems. In: "EUROCAL'87", *Lecture Notes in Computer Science* 378, 81-92, Springer-Verlag, Berlin, 1989.

[7] Gerdt V.P., Zharkov A.Yu. Computer Generation of Necessary Integrability Conditions for Polynomial-Nonlinear Evolution Systems. In: "ISSAC'90" *(International Symposium on Symbolic and Algebraic Computation)*, ACM Press, Addison-Wesley Publishing. Co., 1990, pp. 250-254.

[8] Gerdt V.P., Zharkov A.Yu. Algorithms for Investigating Integrability of Quasilinear Evolution Equations with Non-Degenerated Main Matrix. JINR Report R5-91-255, Dubna, 1991.

[9] Mikhailov A.V., Shabat A.B., Yamilov R.I. Symmetry Approach to Classification of Nonlinear Equations. The Complete List of Integrable Systems. *Usp. Mat. Nauk.* 42, 3-53, 1987 (in Russian).

[10] Buchberger B. Groebner Bases: an Algorithmic Method in Polynomial Ideal Theory. In: (Bose N.K., ed.) Recent Trends in Multidimensional System Theory, Reidel, 1985.

[11] Kredel H., Weispfenning V. Computing Dimension and Independent Sets for Polynomial Ideals, *J. Symb. Comp.* 6, 231-247, 1988.

[12] Gerdt V.P., Khutornoy N.V., Zharkov A.Yu. Solving Algebraic Systems Which Arise as Necessary Integrability Conditions for Polynomial-Nonlinear Evolution Equations. JINR E5-90-48, Dubna, 1990.

[13] Hearn A.C. REDUCE User's Manual. Version 3.3. The Rand Corporation, Santa Monica, 1987.

[14] Manakov S.V. On the Theory of Two-Dimensional Stationary Self-Focusing of Electromagnetic Waves. *JhETF* 65, 2. pp.505-516, 1973 (in Russian).

[15] Schulman E.I. On the Integrability of Long and Short Waves Resonant Interaction Equations. *Dokl. Acad. Nauk. USSR* 259, pp.579-581, 1981 (in Russian).

[16] Khuhkhunashvili V.Z. To the Integrability of the System of Two Nonlinear Schrödinger Equations. *Theor. Math. Phys.*, 79, 2, pp. 180-184, 1989 (in Russian).

Gröbner Bases in Mathematica: Enthusiasm and Frustration.

Bruno Buchberger
RISC-LINZ

—

Research Institute for Symbolic Computation
Johannes Kepler University, A4040 Linz, Austria (Europe)
Tel: +43 (7236) 3231-41, Fax: +43 (7236) 3338-30
Bitnet: k313370@AEARN

—

February 8, 1991

Abstract

We report on an experimental implementation of Gröbner bases in Mathematica. This experiment gives insight into the performance of Mathematica as a scientific system for algorithm researchers. We draw two major conclusions:

- We are enthusiastic about Mathematica with respect to the programming style it supports, which allows easy, well-structured, and generic implementation of algorithmic ideas.
- We are frustrated because Mathematica is so extremely slow that its use for scientific experiments of serious size is prohibitive.

1 The Experiment and Its Result

The experiment described in this paper is part of the book project (Buchberger 1991). The book on which the author is working intends to provide an easy, but mathematically complete, introduction to the theory of Gröbner bases and its many applications. We also intend to distribute, along with the book, a GRÖBNER software package in source code. The goal of GRÖBNER is threefold:

- *students* should be supported in learning the theory by having the possibility to try out examples and to see all details of an implementation,

- *researchers* should be able to experiment with new versions of the algorithms, check hypotheses, expand and improve the package,
- and *users* should be encouraged to apply the method for various concrete large-scale problems.

A suitable language for GRÖBNER should therefore meet the following requirements:

- the language should be *available on as many machines as possible*,
- the language should be *professionally distributed and supported*,
- the code should be *easily readable*,
- some *basic algebraic algorithms* (long integer arithmetic and, in refined versions of the package, rational function arithmetic and polynomial factorization) should already be available in the language,
- the language should support *generic programming* (formulation of algorithms independent of the underlying data domain),
- the *code should be fast.*

I compared systematically a number of available languages that might be considered as an immediate choice: C, C^{++}, LISP, PCL (Portable Common Loops), muSimp, SAC-2, Scratchpad-2, Maple and Mathematica. After extensive experiments with coding all or part of GRÖBNER in these languages, I found that the appropriateness of these languages for the task at hand can be summarized in the following table:

	C	C^{++}	Lisp	PCL	muSIMP	SAC-2	Scratch-pad-2	Maple	Mathe-matica
availability	+	+	+	∓	-	+	-	+	+
professional distribution	+	+	+	∓	+	-	+	+	+
readability	±	±	±	±	+	+	+	+	+
algebraic algorithms	-	-	-	-	±	+	+	+	+
genericity	-	+	-	+	-	-	+	-	+
speed	+	+	+	∓	+	+	∓	-	−

(Maple will soon have generic programming facilities.)

The coarse scale used in this table is: $+, \pm, \mp, -$. The last line concerning speed can be given in more detail:

	C	C^{++}	Lisp	PCL	muSIMP	SAC-2C	Spad-2	Maple	Mathematica
speed	1	1/2	1/10	1/100	1/10	1	1/100	1/1000	1/3000

This line should be understood in the way that, for example, a program written in Mathematica as a language (not a call to an eventually available built-in function) is approximately 3000 (three-thousand!) times as slow as the same program written in C. One main part of this paper (Section 3) will be devoted to backing this assertion.

When I started my experiments with the above candidate languages for GRÖBNER, I already knew most of the entries in the above rough table of performance criteria because most of this is well know and documented. The two entries that surprised me most (and it took me quite some time to "fill these entries in" because very little is said about this in the official documents and critical assessments) were

- the strength of Mathematica for generic programming and
- its prohibitively slow speed.

Therefore, I would like to devote this paper to a discussion of these two aspects.

The conclusions we draw are, of course, independent of the particular package considered. Only for the examples we will need some basic knowledge about Gröbner bases, see the survey article (Buchberger 1985). It is clear that the present paper cannot give an introduction to Mathematica either. For all details about Mathematica see the document (Wolfram 1988).

2 Programming Style and Generic Programming

The programming style of Mathematica is elegant mainly because of two reasons:

- Mathematica's fundamental data type is the "expression", which models both nested data and nested function descriptions and encompasses the expression encounterd in ordinary mathematics in a very natural way.
- In function definitions, Mathematica allows arguments that are "patterns" (i.e. terms) and not only variables. Thus, one often can formulate algorithms in Mathematica without explicit "selectors" and "constructors", which tend to make programs clumsy in other languages. Again, this pattern matching programming style is what normally is used in ordinary mathematics (or in its formalization in predicate logic) for describing algorithms.

For example, any selection of the ordinary mathematical rules for "lim" would immediately yield an executable Mathematica program:

```
Limes[a_+ b_] := Limes[a] + Limes[b]
Limes[a_ b_] := Limes[a]  Limes[b]
```

Every Mathematica expression that matches the "pattern" Limes[a_+ b_], where a_ and b_ stand for arbitrary expressions, would be transformed by the Mathematica interpreter according to the first rule. In fact, Limes[a_+ b_] is an abbreviation for Limes[Plus[a_, b_]]. Even, an entire rule like Limes[a_+ b_] := Limes[a] + Limes[b] is in fact a single expression, which in "full form" would be

```
Define[ Limes[ Plus[ Blank[ a], Blank[b]]],
    Plus[ Limes[ a], Limes[ b]]]
```

Generally speaking, except for some atoms, the only data items in Mathematica are expressions of the form $f[e_1, \ldots, e_n]$, where $f, e_1, \ldots, e_n$ are again expressions.

This simple concept of "expression and matching" is very powerful. We will now show that it incorporates, in a very natural way, the concept of "generic programming", which is vital for complex algebraic packages. We find it worthwhile to expand on this point because, although briefly mentioned in the document (Wolfram 1988), it is not so commonly known.

The main point how generic programming can be incorporated in Mathematica programs is the fact that the f in a Mathematica expression $f[e_1, \ldots, e_n]$ can be used as a "tag" for characterizing a data domain. Objects having two different (constant) tags T1 and T2 will automatically be analyzed to belong to two different domains and, accordingly, two different sets of rules may be installed for the same operation Operation:

```
Operation[ T1[ ...]]   := first right-hand side,
Operation[ T2[ ...]]   := second right-hand side.
```

When finding an expression of the form Operation[*expr*], the Mathematica interpreter analyzes the expression *expr* and depending on whether *expr* starts with T1 or T2 applies the first rule or the second. This simple mechanism can be used to create "generic packages" that handle complicated "towers" of algebraic domains without repetition of code.

For example, when implementing Gröbner bases (over multivariate polynomials) one would like to formulate the algorithms for a wide range of different domains of coefficients and for many different orderings and various representations of power products. Therefore it is not possible to use the built-in multivariate polynomial package of Mathematica (which is actually coded in C and is not available in source code for users!)

One way of organizing the many possible "towers" of domains for GRÖBNER is as follows: The top-most domain is the domain of distributive polynomials which I chose to represent in the following "nested" form:

DNP[*m*, *dnp*],
    where m is a monomial and
    dnp is again a distributive polynomial.
(DNP[ ] is the "empty" (zero) polynomial.)

The monomials might be represented in the following form

Mon[*c*, *pp*],
    where c is an element of a coefficient domain and
    pp is an element of a power product domain.

We are interested in many different coefficient domains: the built-in rational numbers, the finite fields GF(p), the field of rational functions etc. For example, finite field elements in GF(p) might be represented by

FF[*f*],

where f is a number modulo a prime p, and rational functions may be represented by

RF[*rf*],

where *rf* might be a rational function in the built-in Mathematica representation.

Finally, the power products may be represented as "exponent lists" in the form

EL[$e_1, \ldots, e_n$],

where the $e_1, \ldots, e_n$ are the exponents at the n indeterminates. Alternatively, one may be interested in a "Gödel coding" of the exponents $e_1, \ldots, e_n$ by the natural number $p_1^{e_1} \cdots p_n^{e_n}$, where the $p_1, \ldots, p_n, \ldots$ are the prime numbers. A corresponding representation in Mathematica may be

GE[*ge*],

where *ge* is a natural number.

Now we show some parts of the corresponding Mathematica code for realizing arithmetic on these domains:

Addition for DNP*-polynomials*:

```
dnp_DNP + DNP[] := dnp

DNP[ m1_, dnp1_] + DNP[ m2_, dnp2_] :=
    DNP[ m1, dnp1 + DNP[ m2, dnp2]] /; m1 > m2

...

DNP[ m1_, dnp1_] + DNP[ m2_, dnp2_] :=
    DNP[ m1 + m2, dnp1 + dnp2] /; IsEquivalent[ m1, m2] && m1 != -m2
    ...
```

(A rule containing a variable like dnp_DNP on the left-hand side must be read as follows: If an argument dnp whose tag is DNP is encountered then the rule is applied). Note that the "+" on the left-hand side of these rule denotes the addition specific for DNP-polynomials whereas the "+" on the right-hand side is "generic" in the sense that, at run-time, the objects m1, m2, dnp1, dnp2 etc. are analyzed and, in dependence on their "tag" (in this case "Mon" or "DNP"), the appropriate rule of the package is selected and applied. Similarly, in this example, also ">", "–", and "IsEquivalent" are "generic".

Some operations on Mon-*monomials*:

```
Mon[ c1_, pp1_] > Mon[ c2_, pp2_] := pp1 > pp2
Mon[ c1_, pp_] + Mon[ c2_, pp_] := Mon[ c1 + c2, pp]
IsEquivalent[ Mon[ c1_, pp1_], Mon[ c2_, pp_]] := pp1 == pp2
```

The operations ">", "+", and "IsEquvialent" appearing on the left-hand side of these definitions are the specific realizations for Mon-polynomials of the generic operations used in the above domain of DNP-polynomials. The operations ">" and "+" on the right-hand side are again "generic".

Let us consider one more "layer" in this example of a "tower of algebraic domains".

">" on EL-*power product*:

```
el1_EL > el2_EL :=
    Block[ i, ...,
        For[ i = 1, ...
            ...
            If[ el1[[ i]] > el2[[ i]],
                ...
                ]
            ]
        ]
```

">" on GE-*power products*:

```
GE[ e1_> GE[ e2_] := e1 > e2
```

In these two definitions, again, ">" on the left-hand side denotes the operation specific for the domain of EL-power products and GE-power products, respectively. ">" on the right-hand side denotes the corresponding generic operation. Typically, the domain of natural numbers will be used as the substitute for these generic domains. However, it is well conceivable that other domains are used as exponent domains. For example, it may turn out that "symbolic exponents" (i.e. polynomial or rational expressions) are an interesting exponent domain.

Addition on rational number coefficients: Arithmetic on these numbers is built-in. In fact, these numbers have the internal tag `Rational`.

Addition on finite field elements:

```
FF[ f1_] + FF[f2_] := FF[ Modul[ f1 + f2, $Prime]]
```

Addition on built-in rational functions:

```
RF[ rf1_] + RF[ rf2_] := RF[ Factor[ rf1 + rf2]]
```

In the last two examples, again, "+" on the left-hand side is specific for the domains "FF" and "RF", respectively, whereas "+" on the right-hand side is generic. Typically, in these cases, the generic rule will only be applied to the domain of integers and the domain of built-in rational function expressions, respectively.

On top of the DNP-polynomials, a number of higher levels in the generic domain hierarchy are constructed in GRÖBNER, for example, the domain of sets of polynomials. pairs of polynomials, sets of pairs of polynomials etc.

A package that is constructed according to the above principle can then be used for a huge variety of domain combinations. At run-time, the Mathematica pattern matcher will analyze the tags of the data expressions and select the appropriate rules in the rule base (= "program"). Roughly, if in a package with m "layers" of domains there are n domains in each layer then the package can be used for n^m many concrete domains although there are only $m \cdot n$ many pieces of code!

Summarizing, I think that generic programming in Mathematica along the above lines for towers of algebraic domains is elegant, natural, versatile, and yields intelligible. easy-to-change and short code. I really enjoyed programming in this style.

In addition, the slow-down caused by tagging objects is tolerable. By appropriate distribution of the code (how this can be controlled by the programmer is described in the Mathematica document), as a rule of thumb the slow-down is approximately by the factor of 2 if one has 10 rules in each domain.

Having adopted the above generic style for a preliminary version of GRÖBNER I was nearly convinced to stay with Mathematica for carrying the project through. I drastically changed my mind when I started systematic measurements of computing time: It turned out that, whereas the relative slow-down by generic programming is tolerable, the absolute computing times are prohibitive. I report on this in the next section.

3 Speed

Speed is important both for the *user* of a system like GRÖBNER and for the *researcher*.

Algebraic methods like Gröbner bases, which are "universal" for a broad class of problems, tend to be "exponential" in their behavior. High speed is therefore essential for *using* the method in practical cases. For example, good results have been achieved recently in robot kinematics by using the Gröbner bases method on the PSI machine at ICOT (Tokyo), see (Sato, Aiba 1991). The computing times are in the range of several seconds, which is quite tolerable for this kind of problems. A slow-down by a factor of 1000 or even "only" 100 by using the wrong language would make the application worthless.

However, also the *researcher* working in such an area heavily depends on speed because he needs to study huge series of test examples for observing the dependence of computing time on input parameters, for studying certain phenomena in the intermediate results that may lead to new conjectures and eventually to new theorems, and also for using the method as a building block for other algorithmic problems. For example, recently the Gröbner bases method is heavily studied as a building block in the context of Zeilberger's approach to the automated generation and proof of combinatorial identities and and the computation of definite sums and integrals, see (Takayama 1990).

For avoiding misunderstandings, let me point out that Mathematica (and similar systems like MAPLE) may be very fast if one uses the *built-in* C functions. Externally, these functions can be called by Mathematica function calls. Internally, however, they are not written in the Mathematica language but in C. Their code is not accessible and even if it would be accessible it would be of little use for the tutorial and research purposes described in this paper.

Even this favorable statement about the speed of built-in functions in Mathematica must be relativized because some of the functions, in particular the built-in Gröbner bases function, perform fast on small examples but show an unexplained increase in computing time on slightly bigger examples. This is absolutely intolerable for using the system as a research tool. Not only does the unexplained increase in computing time lead to the conclusion that the implementation does not use all theory available for the method but it leaves the researcher with absolutely no possibility to analyze the reason for the unexplained behavior. Also, it is not possible to adjust the built-in functions to changing needs, for example in the case of GRÖBNER, to variable coefficient domains and variable admissible orderings of power products.

In a situation like ours where we want to distribute software in source code for students, researchers and users in fully documented form, the speed of Mathematica must be judged by its performance for algorithms that are fully formulated in Mathematica with any resort to undocumented algorithms written in C. Since my initial experiments with the speed of parts of GRÖBNER written in Mathematica were so disappointing I went through a detailed time analysis of the fundamental Mathematica operations. Summarizing what I found is:

Class of Operations	Time per operation in millisec (on an Apollo 3500)
constant time operations on Mathematica "lists" (i.e. arrays internally): e.g. Length, First, Last, Part	1.5
constant time operations on "nested lists": e.g. FirstN, RestN, PrependN	3
Iteration over Mathematica "lists": e.g. Rest, Drop, Prepend, Append, Insert, Reverse	1 + 0.04 l
Map and Scan iteration over "lists"	1.5 l
user programmed iteration over "lists" using "For" etc.	3 l
user programmed iteration over "nested lists" LengthN, ReverseN, MapN, ScanN etc.	6 l

The parameter "l" in the above table denotes the length of lists. In order to understand the length-dependent complexity of some of the above operations one must know that Mathematica "lists" internally are represented by arrays. (This is nowhere documented in the Mathematica publications. However, for algorithm researchers this is very important information.)

The "nested lists" mentioned in the above table are expressions of the following kind:

```
T[ e1, T[ e2, T[ e3, ...  T[ ] ...]]],
```

where T is some "tag" and the ei are the actual elements of the list. This may be used as one possible simulation of true "lists" in Mathematica. The user-defined operations on such nested lists have the suffix "N" in the above table.

From the above table one sees that the Mathematica operations are approximately 3000 - 8000 times slower than the corresponding operations programmed in C. (The Apollo 3500 is a 5 MIPS machine). (My experimental results are also backed implicitly by the examples in (Maeder 1990), which partly contain timings. However, no explicit mention about speed is made in the official Mathematica documents!)

As a consequence, high-level algorithms fully written in the Mathematica language and not resorting to built-in medium grain algorithms like polynomial arithmetic are prohibitively slow. Here are the computation times of two typical Gröbner bases examples:

	3 variables 3 polynomials degree 2	3 variables 3 polynomials degree 4
My implementation on a ZUSE Z23 (1965!) in assembler code	2 min	12 min
Built-in Mathematica Gröbner basis function on Apollo 3500	1 sec	> 2 days
Built-in Maple Gröbner basis function on Apollo	4 sec	2 min
My implementation fully coded in Mathematica language	5 min	30 min

From this table one sees that the speed-up of approximately 1000 achieved by hardware improvements in the last 25 years (the ZUSE Z23 was a 0,003 "MIPS" machine!) is completely lost by the software elegance of Mathematica and similar high-level languages.

Also, one sees that the undocumented built-in Gröbner bases function of Mathematica performs very well on small examples but shows an unexplained increase in performance for slightly larger examples. (This phenomenon was observed by many researchers with some other of the built-in Mathematica functions).

The reasons for this bad speed performance of Mathematica as a language most probably are manifold:

- Mathematica is interpreted, not compiled,
- a function call in Mathematica, since it performs something so general as pattern matching, needs approximately 1 millisecond independent of the function body,
- in integer arithmetic the most general case is assumed even if only index computations are executed,
- others?

I think that these reasons should be analyzed in more detail and documented in future Mathematica documents.

4 Conclusions

In this paper I concentrated on only two criteria for judging Mathematica. Many other criteria could be considered, see the extensive critical analysis (Fateman 1990). Interestingly enough, in (Fateman 1990) the two criteria, on which I concentrate in this paper and which are the crucial ones for judging a system as an instrument for algorithm research, are not treated in any detail.

From what I learned in the above experiments I drew the following conclusions:

- Although there are many interesting software systems available for computer algebra, the "ideal" system does not yet exist. The ideal system would combine the elegance and naturalness of Mathematic and the speed of C. I know this is impossible but I think we could achieve something much better than what exists now. For example, it would already help a lot if there was a possibility to incorporate user-defined C routines in high-level languages like Mathematica.

- For my own GRÖBNER project I now decided to use Collins' SAC-2 system in a new version that is entirely coded in C. I designed a simple preprocessing mechanism that allows a rudimentary form of generic programming that seems to be sufficient for a project like GRÖBNER. I will report on this in a subsequent paper. G. Collins and I decided to cooperate on turning "SAC-2C" into a professionally distributed system such that many researchers can use it in situations like GRÖBNER where algorithm researchers want to distribute easy-to-read, generic and fast source code.

Acknowledgment: This paper was written in the frame of the "Gröbner Bases Project" sponsored by the Austrian Ministry of Science and Research. I would also like to thank Roman E. Maeder for pointing out the possibility of generic programming in Mathematica to me. Thanks also to Stephen Wolfram who provided a test installation of Mathematica for the "Gröbner Bases Project".

References

B. Buchberger. Gröbner bases: an algorithmic method in polynomial ideal theory. In: *Multidimensional Systems Theory* (N. K. Bose editor), pp. 184 - 232. D. Reidel Publishing Company, Dordrecht - Boston - Lancaster, 1985.

B. Buchberger. *Gröbner Bases: Elementary Theory and Applications.* Springer-Verlag, New York - Vienna, in preparation.

R. Fateman. *A Review of Mathematica.* Manuscript. Dept. of Computer Science, University of California, Berkeley, 1990.

R. E. Maeder. *Programming in Mathematica.* Addison - Wesley Publishing Company. Redwood City, California, 1988.

S. Sato, A. Aiba. *An Application of CAL to Robotics.* Manuscript, ICOT Institute, Tokyo, 1991.

N. Takayama. *An Approach to the Zero Recognition Problem by Buchberger's Algorithm.* Dept. of Mathematics, Kobe University, submitted to publication.

S. Wolfram. *Mathematica: A System for Doing Mathematics by Computer.* Addison - Wesley Publishing Company, Redwood City, California, 1988.

Examples of Problem Solving using Computer Algebra

Dominique DUVAL [*]
Laboratoire de théorie des nombres et algorithmique
Université de Limoges

January 1991

IFIP Working Conference
on Programming Environments for High-Level Scientific Problem Solving
September 23-27, 1991, Karlsruhe

Introduction

Computer algebra, in contrast with numerical analysis, aims at returning exact solutions to given problems. One consequence is that the shape of the solutions may, at first, look somewhat surprising.

In the first two sections, we present two examples of problem solving using computer algebra, with emphasis on the shape of the solutions. The first example is the resolution of linear differential equations with polynomial coefficients, and the second one is the resolution of polynomial equations in one variable. In the first example the solution may look useless since it makes use of divergent series, and in the second example the solution may look rather awkward. But in both examples we sho that these solutions are in the right shape for a lot of applications, including numerical ones. In the third section we show that some features of the computer algebra system Scratchpad, especially strong typing and genericity, are useful for the implementation of a method for our second problem, *i.e.* for the implementation of the "dynamic" algebraic closure of a field.

[*] Also supported by Greco de calcul formel and PRC mathématiques-informatique

1 Linear Differential Equations

This section describes part of the work made by J. Della Dora, J.-P. Ramis and their groups at the universities of Grenoble and Strasbourg. Using the computer algebra system Reduce, they wrote a package called DESIR for the resolution of linear ordinary differential equations with polynomial coefficients, like *Airy equation*

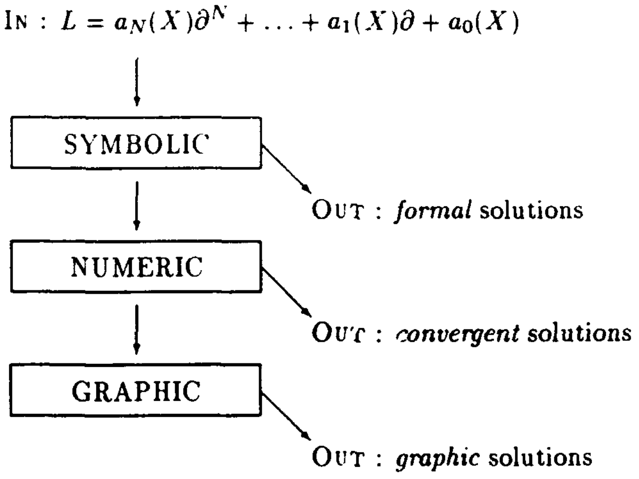
$$y'' - x = 0 \tag{1}$$

or more generally $L(y) = 0$ where L is some linear differential operator

$$L = a_N(x)\partial^N + \ldots + a_1(x)\partial + a_0(x)$$

so that $L(y) = a_N(x)y^{(N)} + \ldots + a_1(x)y' + a_0(x)y$. In DESIR, the *resolution* of the equation $L(y) = 0$ has three different meanings :

1. A *symbolic* solver returns N independent *formal* solutions of $L(y) = 0$ at $x = 0$.
2. Using the formal solutions, a *numeric* solver returns N independent *convergent* solutions of $L(y) = 0$ at $x = 0$.
3. These solutions in turn are used by a *graphic* solver to get N *graphic* solutions, *i.e.* mappings from the set **C** of complex numbers into itself.


IN : $L = a_N(X)\partial^N + \ldots + a_1(X)\partial + a_0(X)$
SYMBOLIC
OUT : *formal* solutions
NUMERIC
OUT : *convergent* solutions
GRAPHIC
OUT : *graphic* solutions

A description of DESIR is made in the theses [Tournier,Richard], and we refer to them for a detailed description of the algorithms involved, for proofs, and for more examples. We only give here the example of the differential operator

$$A = x^5\partial^2 + 2x^4\partial - 1 \tag{2}$$

of order $N = 2$. The differential equation $A(y) = 0$ corresponds to Airy equation at infinity, *i.e.* it is got from Airy equation (1) by changing x into $1/x$.

1. The symbolic solver returns two independent solutions

$$\begin{cases} x^{\frac{1}{4}} \exp\left(\frac{2}{3x^{\frac{3}{2}}}\right) \sum_{i=0}^{+\infty} c_i x^{\frac{3i}{2}} \\ x^{\frac{1}{4}} \exp\left(\frac{-2}{3x^{\frac{3}{2}}}\right) \sum_{i=0}^{+\infty} d_i x^{\frac{3i}{2}} \end{cases}$$

   where $d_i = (-1)^i c_i$ and

$$c_0 = 1 \ , \ c_1 = \frac{5}{48} \ , \ c_2 = \frac{385}{4608} \ , \ \ldots$$

   and more generally a recurrence relation is given that allows the exact computation of c_i for every i.

2. The series

$$\hat{u}(t) = \sum_{i=0}^{+\infty} d_i t^i = -1 + \frac{5}{48}t - \frac{385}{4608}t^2 + \ldots$$

   is divergent for every $t \neq 0$, but it is the asymptotic expansion (in the sens of "Gevrey of order 2", as explained in [Richard]) of some holomorphic function $u(t)$ for $arg(t) \in \ -\frac{3\pi}{2}, +\frac{3\pi}{2}U$. This function $u(t)$ may be computed by "resummation" techniques, using Borel transform [Richard]. Now let

$$U(x) = x^{\frac{1}{4}} \exp\left(\frac{-2}{3x^{\frac{3}{2}}}\right) u(x^{\frac{3}{2}})$$

   then $U(x)$ is a numerical solution of equation $A(y) = 0$ for $arg(x) \in \ -\pi, +\pi[$. It may be proved that $\frac{1}{2\sqrt{\pi}}U(\frac{1}{x})$ is the usual *Airy function*

$$Ai(x) = \frac{1}{2\pi} \int_{\exp(\frac{-i\pi}{3})\infty}^{\exp(\frac{i\pi}{3})\infty} \exp\left(-xt + \frac{t^3}{3}\right) \, dt$$

   solution of Airy equation (1). Similar resummation techniques give two independent numerical solutions in different parts of the complex planes.

3. Finally, using these numerical solutions it becomes possible to represent graphically any solution of equation $A(y) = 0$, as a mapping from the set of complex numbers **C** to itself. The planar representation chosen in DESIR makes use of time and color to replace the missing dimensions. More precisely, a graphic representation of a numerical solution f (*e.g.* $f(x) = Ai(1/x)$) is made of a succession of images that represent, for increasing values of ρ, the image of the circle $3x3 = 1/\rho$ by f. Each point x on the circle is colored according to its argument, and its image $f(x)$ is colored the same way.

So, the DESIR package proves that it is possible to use the information given by a divergent series to get practical information about the solutions of a differential equation. Actually, the formal solutions are computed at singular points of the differential equation, and these points concentrate a lot of information about the equation.

The singular points of a differential equation $L(y) = 0$ are in finite number : The point at infinity may be singular, and also the roots of the polynomial $a_N(x)$ (in the case of operator A above, $a_N(x) = x^5$, which has $x = 0$ as unique root). Here appears the need of computing exactly with complex numbers that are defined as the roots of a polynomial with rational coefficients. For example if $a_N(x) = x^2 - 2$ the singular points of L are $\sqrt{2}$ and $-\sqrt{2}$. Every numerical value for $\sqrt{2}$, as precise as it may be, corresponds to a non-singular point, where we cannot get so much information about the differential equation. We have to compute precisely with the real numbers $\sqrt{2}$ and $-\sqrt{2}$, not with some floating point approximation.

As an example, let us now consider the differential operator

$$B = a_2(x)\partial^2 + a_1(x)\partial + a_0(x) \tag{3}$$

where

$$\begin{cases} a_2(x) &=& x^6 + 2x^4 + 2x^3 + x^2 + 2x + 1 = (x^3 + x + 1)^2 \\ a_1(x) &=& x^4 + x^3 + x^2 + 2x + 1 \\ a_0(x) &=& x \end{cases}$$

The finite singular points of (3) are the roots of the polynomial $a_2(x)$, *i.e.* the 3 roots of $x^3 + x + 1$. Let α denote *any* root of $x^3 + x + 1$ in $\mathbf{C}$. The symbolic part of DESIR then returns 2 independent formal solutions of equation $B(y) = 0$ at α :

$$\begin{aligned} y_{\alpha,\beta} &= (x - \alpha)^\beta \times \\ &\left(1 + \frac{1}{341}\left((80\alpha^2 - 27\alpha + 136)\beta + (-97\alpha^2 + 37\alpha - 199)\right)(x - \alpha) + \dots\right) \end{aligned}$$

where β is *any* one of the 2 roots of the polynomial (in z, with coefficients depending on α)

$$(-3\alpha^2 - 9\alpha + 1)z^2 + (6\alpha^2 + 7\alpha - 3)z + \alpha\ .$$

2 Algebraic Numbers

Complex numbers that are roots of a polynomial with rational coefficients are called *algebraic numbers*. For example the complex numbers denoted by α in the last section are algebraic numbers. It may be proved that complex numbers that are roots of a polynomial with algebraic coefficients are algebraic numbers too, for example the complex numbers denoted by β in the last section.

So that to solve a polynomial equation in one variable over the set $\mathbf{Q}$ of rational numbers, say

$$P(x) = 0 \tag{4}$$

is to determine the set of algebraic numbers

$$\{\alpha \in \mathbf{C} \mid P(\alpha) = 0\}\ .$$

And actually we are going to show that, very often, the best way to determine the solutions of equation (4) is to say that they are the complex numbers α such that $P(\alpha) = 0$.

2.1 The irreducible case

Let us first prove that it is certainly the best way when :

1. $P(x)$ is irreducible over the rational numbers, *i.e.* it cannot be written as $P(x) = P_1(x)P_2(x)$ with both $P_1(x)$ and $P_2(x)$ non-constant polynomials with rational coefficients,

2. and the only operations we want to perform on the solutions are the four operations $+$, $-$, $\times$ and $/$, and equality tests. In particular we do not ask whether they are real or not, and even in case they are real we do not use any comparison test (like $<$). Let us call this the *unordered case.*

It is the case of the singular points of the differential operator (3) above. The first assumption is satisfied by $P(x) = x^2 - 2$ and $P(x) = x^2 + 1$, but not by $P(x) = x^2 - 1 = (x-1)(x+1)$. The proof of our assertion is the following :

- Computations with one root α of $P(x)$ come to elementary computations on polynomials with rational coefficients modulo $P(x)$ (more precisely, the field $\mathbf{Q}(\alpha)$ is isomorphic to the quotient $\mathbf{Q}[x]/(P(x))$)

- And these computations do not depend on the choice of α (this comes from the fact that $\mathbf{Q}[x]/(P(x))$ does not depend on α).

In the case of $P(x) = x^2 - 2$, this means that if you compute first with $\alpha = \sqrt{2}$ and then with $\alpha = -\sqrt{2}$, you will repeat exactly the same computations, so that you would better compute with one symbol α, using the fact that $\alpha^2 = 2$. For example

$$\begin{aligned}(1+2\alpha)^2 - (1+\alpha)^2 &= (1+4\alpha+4\alpha^2) - (1+2\alpha+\alpha^2)\\ &= 6+2\alpha\end{aligned}$$

which means that

$$(1+2\sqrt{2})^2 - (1+\sqrt{2})^2 = 6+2\sqrt{2} \text{ AND } (1-2\sqrt{2})^2 - (1-\sqrt{2})^2 = 6-2\sqrt{2}\,.$$

In the case of $P(x) = x^2 + 1$, it corresponds to the fact that we *cannot* distinguish between both roots of $P(x)$ (although we know that once one is *called* i, the other one is *equal to* $-i$).

2.2 The reducible case

Let us now forget the first assumption above, *i.e.* $P(x)$ may be reducible over the rationals, but we still are in the unordered case. For example if α denotes any root of $P(x) = x^2 - 1$ (*i.e.* $\alpha = 1$ or $\alpha = -1$) then

$$\begin{aligned}(1+2\alpha)^2 - (1+\alpha)^2 &= (1+4\alpha+4\alpha^2) - (1+2\alpha+\alpha^2)\\ &= 3+2\alpha\end{aligned}$$

for both values of α, which means that

$$(1+2)^2 - (1+1)^2 = 5 \text{ AND } (1-2)^2 - (1-1)^2 = 1\,.$$

This proves that at least some computations may be performed exactly as before, without any need to distinguish between the different roots of $P(x)$, and indeed it is the case for computations that do not involve any equality test. But it is false for some computations involving equality tests, for example (with $P(x) = x^2 - 1$)

$$\mathrm{J}\textit{if}\ \alpha^2 + \alpha = 2\ \mathrm{J}\textit{then}\ \ \alpha\ \mathrm{J}\textit{else}\ \ \alpha + 1$$

since here the answer to the test $\alpha^2 + \alpha = 2$ is *true*J if $\alpha = 1$ and *false* if $\alpha = -1$.

The "mathematical" method here factorizes $P(x)$ over the rationals and considers each irreducible factor in its turn. When $P(x) = x^2 - 1$ it means that we consider first $\alpha = 1$ and then $\alpha = -1$. When $P(x) = x^3 - 1$ (that factorizes as $P(x) = (x-1)(x^2+x+1)$) it means that we consider first $\alpha = 1$ and then any number α such that $\alpha^2 + \alpha + 1 = 0$. This method is algorithmic, since there are algorithms for factorizing polynomials over the rationals [Loos,Lenstra/Lenstra/Lovacz].

With this method, the points with vertical tangent on the elliptic curve $Y^2 = X^3 - X$ are described as the points $(-1,0)$, $(0,0)$ and $(1,0)$.

Another example is the parametrization of the four branches through the point $(0,0)$ of the curve $F(x,y) = 0$, where :

$$\begin{aligned} &F(X,Y) = \\ &\quad Y^{16} - 4Y^{12}X^6 - 4Y^{11}X^8 + Y^{10}X^{10} + 6Y^8X^{12} + 8Y^7X^{14} + 14Y^6X^{16} \\ &\quad +4Y^5X^{18} + Y^4X^{20} - 4Y^4X^{18} - 4Y^3X^{20} + Y^2X^{22} + X^{24}\,. \end{aligned}$$

Parametrizations are given by :

$$\begin{cases} (x(t) = -64\beta_1 t^4\ ,\ y(t) = -64t^4 - 64t^7 + \ldots) \\ (x(t) = -64\beta_2 t^4\ ,\ y(t) = -64t^4 + 64t^7 + \ldots) \end{cases}$$

where β_1 is any root of $16X^2 - 4X + 1$ and β_2 any root of $16X^2 + 4X + 1$, *i.e.* $\beta_1 = (1 \pm 2i\sqrt{3})/8$ and $\beta_2 = (-1 \pm 2i\sqrt{3})/8$.

In contrast, the "dynamic" (or "lazy") method considers α as a "parameter", submitted to the "constraint" $P(\alpha) = 0$, computes as much as possible with this information, and *splits* the problem in several parts when it becomes necessary [Dicrescenzo/Duval 1]. Here no factorization algorithm is required, but only gcd computations, which are simpler to implement, faster to run, and valid over every base field (here we only care about the base field **Q**, but in practice a lot of other fields are important for applications). However the implementation is not easy, because of the splittings :

- At every moment there is a "canonical" representation of the algebraic numbers, but it evolves during computations.
- When one case splits into several ones at run-time, from the splitting point the subcases must be treated "in parallel".

On the other hand, the "dynamic" method generalizes to various situations where "parameters" are useful.

Now the points with vertical tangent on the elliptic curve $Y^2 = X^3 - X$ are described as the points $(\alpha, 0)$ where α is any complex number such that $\alpha^3 - \alpha = 0$.

The parametrizations of the four branches of the curve $F(x,y) = 0$ at $(0,0)$ are now described as :

$$(x(t) = -64\beta t^4 \ , \ y(t) = -64t^4 - 64\alpha t^7 + \ldots)$$

where α is any number such that $\alpha^2 - 1 = 0$ and, for each choice of α, β is any number such that $16\beta^2 - 4\alpha\beta + 1 = 0$.

3 Scratchpad Implementation

The second method described above has been implemented by C. Dicrescenzo and the author first in Reduce [Dicrescenzo/Duval 1], and then in a much more general setting in Scratchpad [Jenks,Dicrescenzo/Duval 2].

Actually in the Reduce package everything was written in the underlying Lisp, since we wanted to master the representation of the algebraic numbers by univariate polynomials. This package was only able to handle algebraic numbers over the base field **Q** (*i.e.* algebraic numbers in their "strict" meaning, as defined above).

In Scratchpad it was easy to build a polynomial domain adapted to our representation needs. In addition any "computable" field may be used as a base field. More precisely, we have defined in Scratchpad the *dynamic algebraic closure* of any field. Remember that, by definition, "the" algebraic closure of a field **K** is a field $\overline{\mathbf{K}}$ such that :

1. $\overline{\mathbf{K}}$ contains **K**,
2. every non-constant polynomial with coefficients in $\overline{\mathbf{K}}$ has a root in $\overline{\mathbf{K}}$,
3. and $\overline{\mathbf{K}}$ is a small as possible for these properties.

For example, the field of complex numbers is the algebraic closure of the field of real numbers, but it is not the algebraic closure of the field of rational numbers because it does not satisfy axiom (3) above. In the Scratchpad version, axiom (2) is replaced by the more "effective" axiom :

2′. $\overline{\mathbf{K}}$ is endowed with an application which associates to every non-constant polynomial $P(x)$ with coefficients in $\overline{\mathbf{K}}$ an element α of $\overline{\mathbf{K}}$ such that $P(\alpha) = 0$.

So that our Scratchpad program defines a ***domain constructor*** called *DynamicAlgebraicClosure* with one argument K that must be in the *Field* category of Scratchpad, which means that K is a "computable" field (in some reasonable sense). The domain *DynamicAlgebraicClosure(K)* is also in the *Field* category, and in addition there is a function *rootOf* with two arguments : a univariate polynomial P with coefficients in *DynamicAlgebraicClosure(K)*, and an expression a. The value of *rootOf(P,a)* is an element α of *DynamicAlgebraicClosure(K)* such that $P(\alpha) = 0$. The expression a is the "name" used in outputs by Scratchpad for the number α. Finally it must be known that if $f : D_1 \rightarrow D_2$ is any Scratchpad function that makes use of *DynamicAlgebraicClosure(K)*, and if we want to apply f to some element x of D_1, instead of calling $f(x)$ as usual we must call *allCases(f,x)* in order to be sure to get every possible case. The function *allCases* manages the tree generated by the successive splittings during the computation, it may be viewed as an ingenuous infering engine.

Here is an example of a simple application of our package. We define a function *test*, with argument an integer and value a boolean, that uses *DynamicAlgebraicClosure(RationalNumber)*, where *RationalNumber* is the domain of rational numbers in Scratchpad. We then ask for the value of *test* at 0 in every possible case :

```
CL:= DynamicAlgebraicClosure(RationalNumber)
POLY:= UnivariatePolynomial(x,CL)
test(n:Integer):Boolean ==
  p:POLY:= x*(x-1)*(x+1)**2
  a:CL:= rootOf(p,"a")
  q:POLY:= (x-2)*(x-a)
  b:CL:= rootOf(q,"b")
  b = a+1
allCases(test,0)
```

There is Scratchpad answer :

[value is false in case $a = -1$ and $b^2 - b - 2 = 0$,

value is true in case $a = 1$ and $b = 2$,

value is false in case $a = 1$ and $b = 1$,

value is false in case $a = 0$ and $b^2 - 2b = 0$]

It is important to note that here we make use of :

1. Scratchpad facilities for the definition of recursive domains, *since the elements of DynamicAlgebraicClosure(K)* are represented by polynomials with coefficients in *DynamicAlgebraicClosure(K)*.

2. Scratchpad *genericity* properties, since the argument of *DynamicAlgebraicClosure* is any domain K in the *Field* category of Scratchpad (which means any "computable" field, in a reasonable sense),

3. Scratchpad *strong-typing* properties, since *DynamicAlgebraicClosure(K)* is also in the *Field* category, while it is represented by various rings that usually are not fields.

Actually it seems that people agree to say that the genericity properties of Sratchpad are very useful, but that the opinions are more divided about strong-typing, which makes sometimes the use of Scratchpad rather heavy. With our package we prove that strong-typing may be of great interest. Actually the domain *DynamicAlgebraicClosure(K)* may be used everywhere Scratchpad asks for a field, so that every code that has been written in Scratchpad for polynomials, matrices, and so on, may be used with algebraic coefficients. In [Duval/Rybowicz] we give an example of the use of *DynamicAlgebraicClosure(DynamicAlgebraicClosure(K))* in order to study the singular points of curves defined by polynomials with algebraic coefficients.

Conclusion

Symbolic, numeric and graphic tools begin working together for solving problems in the scientific computation area. It is still far from clear to determine the best way to integrate them in order to solve a given problem. One point is that the shape of the solutions may be fairly different, as we have seen on some examples above. We have tried to show on examples that the solutions returned by computer algebra, how "strange" they may look to numericians, may indeed prove very useful, thanks to their exactness and their ability to deal with parameters.

References

[Dicrescenzo/Duval 1] C. Dicrescenzo, D. Duval. *Algebraic computation on algebraic numbers.* Informatique et calcul, Wiley-Masson, p. 54–61 (1986).

[Dicrescenzo/Duval 2] C. Dicrescenzo, D. Duval. *Algebraic extensions and algebraic closure in Scratchpad.* Symbolic and Algebraic Computation, Lecture Notes in Computer Science **358**, Springer, p. 440–446 (1989).

[Duval] D. Duval. *Nombres algébriques et calcul formel.* Bulletin de l'INRIA **130**, p. 26–28 (1991).

[Duval/Rybowicz] D. Duval, M. Rybowicz. *Evaluation dynamique en Scratchpad.* To be published in Calsyf, ed. M. Mignotte, Université de Strasbourg.

[Jenks] R.D. Jenks. *A primer: 11 keys to new Scratchpad.* Lecture Notes in Computer Science **174**, Springer, p. 123–147 (1984).

[Lenstra/Lenstra/Lovacz] A.K. Lenstra, H.W. Lenstra, L. Lovasz. *Factoring polynomials with rational coefficients.* Math. Annalen **261**, p. 513–534 (1982).

[Loos] R. Loos. *Computing in algebraic extensions.* Computer Algebra, Symbolic and Algebraic Computation, 2nd. ed., Springer, p. 173–188 (1982).

[Richard] F. Richard-Jung. *Représentation graphique de solutions d'équations différentielles dans le champ complexe.* Thèse, Université de Strasbourg 1 (1988).

[Tournier] E. Tournier. *Solutions formelles d'équations différentielles. Le logiciel de calcul formel DESIR, étude théorique et réalisation.* Thèse, Université de Grenoble 1 (1987).

Environments for Large-Scale Scientific Computation

Stuart I. Feldman

Bellcore
Morristown, New Jersey, USA

1. Introduction

This paper addresses support for writing large scientific software, measured either in the size of the program or in the resource demands of the execution. Traditionally, scientific programs have been of moderate size by industrial software standards, but extremely demanding of execution resources (processor cycles, primary memory, secondary memory, input/output bandwidth). Large codes are now big enough to demand the same support as other complex software, while the operating regime requires very special care. This paper will discuss the needs of large-scale scientific software development, and note what seems special about this type of program.

Much mathematical and scientific software has been written on the research model: rapid exploration of alternatives and algorithms by a small group of people (frequently one person) with no plans for traceability, evolution, support, or generality. Frequently, the "requirements" are just a set of equations and boundary conditions. Only after the mathematical core of the problem has been solved are the complexities of human interface, portability, reliability addressed. It is only after a program is a success that the deficiencies matter. Many routines and even entire systems have been rewritten in order to achieve portability or to increase the comprehensibility of the final code.

Large commercial software systems, on the other hand, are usually characterized by well-defined phases, including requirements and specifications, design, implementation, test, integration, and release*. Writing software that will have a long life and perhaps involve a large team of developers, designers, and testers demands care and deserves mechanized support.

2. Large Scale Software Development

The easiest measure of size is the number of lines of code. Though one can Stretch these numbers, and more subtle metrics are available, but with careful definition (and a formatter to make the coding style uniform) it can be a useful measure. Really large systems in the telecommunications, defense, and commercial worlds are likely to be measured in the millions of lines. The biggest suites of programs exceed ten million lines of code. This should be contrasted with the size of a subroutine in a mathematics library (a few hundred lines), the size of a mature scientific library (a half million lines), or the size of a very large scientific code such as used for structural analysis (half million to a million lines). These numbers suggest that scientific programs are not (yet) as large as the worst cases in other fields. The complexity and amount of analysis behind the best scientific programs, however, makes it fair to compare them to operating systems code, where we are again in the realm of hundreds of thousands rather than multiple millions of lines. These mega-projects also employ hundreds of people for many years.

* Even though these terms refer most directly to the much maligned waterfall model of software development, they are still useful for iterative models involving rapid prototyping or spiraling.

Although some scientific codes appear to be eternal, only occasionally are really large teams involved. According to one specialist, the typical experience with a big problem is that the prototype and proof of concept takes a few weeks, the full implementation takes most of a year, and then the next few years are spent exploring the underlying mathematics and physics. Debugging and testing happen at each stage; what is being sought changes.

3. Software Development and Environments

Large scientific programs require at least as much support as other kinds of software. It is unfortunate that scientific programmers do not always take advantage of tools that would simplify standard problems.

Each phase of the life cycle makes use of specific tools and techniques: requirements analyzers in the early stages, compilers and debuggers in the implementation phase, regression test tools for integration testing. In addition to this task-specific software, there are also needs that cross multiple phases. Furthermore, support is needed to help coordinate the activity of teams of people and machines. When enough pieces are available and share data and interfaces, we can speak of an integrated software development environment rather than a collection of disconnected, albeit useful tools. Some examples of desired environment services are:

3.1. Software Data

It is essential to store a wide variety of objects relating to the software development, and to be able to associate and retrieve them later. These include not only code, but also documents, requirements, design reviews, test sets, and many other types of information. Carefully controlled file systems are the usual implementation route, but they do not provide ideal support.

Closely related to the database issue is maintenance of multiple versions, either because of historical evolution (bug fixes, functionality upgrades) or parallel usefulness (suitability for differing machine architectures or utilizing different operating systems).

Configuration management software provides a way to combine many pieces in a variety of forms. These constellations should be reproducible and modifiable.

Controlled data sharing and transactions provide means of coordinating operations and maintaining consistency.

3.2. User Interface Software

It is a commonplace that the software associated with the human interface is an order of magnitude larger than the computational core of a system. Modern toolkits (such as those utilizing the X window system) are usually themselves many hundreds of thousands of lines. They provide many services, including access to programs, good graphics and uniform control of user input and output devices (such as screens, speakers, mice, and keyboards).

The user interface software also assists the coordination among developers and helps maintain a consistent image of the development process.

3.3. Process Related Software

The operations to be performed when building software can be very long and complex. Mechanical assistance is essential to avoid errors and to improve efficiency. The assistance provided can range from simple collections of commands to knowledge-based assistants. Between these extremes are very useful programmable build tools and explicitly controllable process models.

In order to maintain control over the various concurrent activities, and to be able to replicate and modify these operations, it is necessary to remember what was done and to record measures of the activity (resources used, failures caused, and so forth). Gathering of such metric data is highly desirable if not very common in most environments.

These process control and measurement programs must operate across all phases of the life cycle in order to do the most good. Distributed control and agents are essential for team coordination.

4. Needs of Scientific Computing

These comments apply to almost any type of software development. What is special about mathematical or scientific computing? The following is a list of the most telling differences, and their implications for the ideal environment

4.1. Graphics

Almost all modem scientific applications make very heavy use of graphics to plot functions, indicate boundaries and contours, and so forth. A good picture is indeed worth a thousand numbers. Scientific applications pioneered the use of color displays and graphics libraries. It is common for the display activity to take more computing and more memory than the main calculation, and for this to be a proper use of resources. The graphic display software must be properly integrated and easily accessible to be most useful: it is useful to be able to plot graphs based on symbolic formulas, large numeric data sets, and fresh inputs. Graphics are essential to development and testing, not just the final execution. For example, it is almost impossible to correct huge computations without graphical support in the debugger.

4.2. Multiple Processors

The computing demands of scientific problems appear insatiable, and most efforts at building parallel or distributed computers have been driven by scientific problems first, others later. It is common to use a massive machine to crunch the numbers and then satellite machines to handle human interfaces and analysis. A few processors sharing memory, or a multitude communicating explicitly, usually need to be programmed in different ways. Problems of synchronization, data sharing, and so forth are thus intrinsic to heavy scientific computing. For development, it is necessary to provide debugging facilities that permit single thread executions when necessary, and otherwise help to untangle multiple simultaneous executions. The possibility of nondeterminism makes for many interesting problems.

4.3. Input/Output Requirements

High-resolution graphics already put a significant I/O load on applications. Problems that manipulate huge amounts of data, frequently streamed through the processors, require special systems architectures and applications approaches. Disk striping and parallel arms were first introduced for scientific problems. The old rule of thumb of a megabyte/second of I/O capacity for each MIPS suggests very real problems are coming with massively parallel multi-gigaflop computers.

4.4. Data Storage and Manipulation

Big scientific problems consume and generate huge amounts of data, much of which must be stored. It is not uncommon for a big project to need gigabytes of storage even during the early phase of development. Huge data resources are also common in the commercial world, but some types of data (e.g., floating point numbers, formulas and equations, graphss) and the need for rapid access to ordered data sets are particular to scientific computing.

4.5. Floating Point

Scientific computing is usually equated to the use of floating point. Although this is not necessary, it is usually the case. Whenever real arithmetic appears, complex and varying precision floating point also can be expected. Issues of precision choice, overflow, and so forth are part of the art of mathematical computation. The environment must not only support floating point, but provide special access to techniques such as interval arithmetic and special data representations.

4.6. Symbolic Computation

Scientific computing is not only numerical. All numerical problems begin with a symbolic representation in terms of formulas, equations, and so forth. Some problems are best solved purely in the analytic domain, many benefit from a mixed approach, with parts of the problem handled by symbolic manipulation, followed by numerical approximations, followed perhaps by another analytic phase. Symbolic calculations will be extremely common during the development phase, and still important in the middle of the computation.

4.7. Exception Handling

Although handling of exceptional conditions is a general software design and implementation problem, it is particularly acute with scientific computing because of the many ways a floating point computation can go astray. We already have problems with imprecise interrupts on scalar machines; the problems are far more complex with optimized array or vector code. The handling of exceptions is partly a hardware problem, partly a systems architecture problem, partly a language problem, and partly a user interface problem. No single solution will suffice.

4.8. Languages

Scientific computing has for many years been done primarily in dialects of Fortran. Many special purpose languages have been devised to handle specific tasks, and many of the tools needed to build sophisticated programs are not available in Fortran. Many other languages are now also used, but the Fortrans probably dominate. (Algebra systems are frequently written in LISP, windowing packages are likely to be written in C.) Thus, multiple language environments, with the complexities this forces on interface checking and on communication, are a fact of life for large-scale scientific computing.

4.9. Libraries

Much of the reuse and communication of algorithms in the fields of scientific computing have traditionally been handled through libraries, either locally maintained or internationally available. This dependence on libraries has many advantages, but it does force certain assumptions about separable modules. In the future, library services will continue to be very important, but as mentioned elsewhere closed packages will also be very important elements of the construction of big systems.

5. Future Environment Issues

In the future, scientific programs will be assembled out of larger units – subsystems and packages rather than routines from libraries. This means that the development environment must make it easier to assemble the right versions and test them. The programs are likely to be larger, putting more demands on the execution capabilities of the development environment. As parts will be proprietary, source code will frequently not be available, and so the debugging techniques will need to be able to help even in the face of enforced ignorance.

As massively parallel machines become more common, the development environments will need to have parallel capabilities to provide realistic testing and debugging facilities. (It is not acceptable that the program under test take a century to execute, and it must suffer the same synchronization problems as the full-blown version.)

6. Acknowledgement

I thank W. Kahan for illuminating discussions.

SOFTWARE TOOLS FOR PARALLEL PROGRAM DEVELOPMENT *

Hans P. Zima and Barbara M. Chapman
Department of Statistics and Computer Science
University of Vienna
Rathausstrasse 19/3
A-1010 Vienna AUSTRIA

Abstract

This paper describes restructuring systems for the development of parallel programs to run on a variety of different architectures. The authors explain why they believe that current methods must be supplemented by knowledge-based techniques if such tools are to mature to the extent that they are able to provide efficient and powerful support environments for numerical programming on parallel machines, where performance of the target code is crucial.

Design criteria are introduced for a multi-layer, multi-target, intelligent programming environment for the efficient solution of numerical problems on a variety of parallel architectures. At the highest level, this environment permits a numerical problem formulation in a problem-oriented specification language, using familiar mathematical concepts such as PDEs or ODEs. The system provides a rich set of automatic tools which transform specifications step by step into architecture-specific Fortran dialects that exploit the inherent parallelism of the given architecture. Fortran 90, the new Fortran standard, plays a central role in the environment by serving as a target as well as a source language.

1 Introduction

Parallel programs are very much harder to develop, debug, maintain, and understand than their sequential counterparts. One reason is the difficulty to establish correctness - which must take into account temporal conditions such as liveness, deadlock-freeness, process synchronization and communication. Another reason is the diversity of concurrent architectures and the need to produce a highly efficient program, fine-tuned to the specific target architecture. The impact of task granularity on a concurrent algorithm, the properties of the memory hierarchy, and the intricacies involved in the exploitation of multi-level concurrency, for example, must all be carefully analyzed and used to tune a program. The adaptation of an initially inefficient algorithm to a specific hardware is often called performance debugging, a term that suggests that the correctness criteria for a concurrent algorithm should include requirements for its performance on a given architecture. An inefficient, but otherwise correct program is of practically no use for execution on a supercomputer.

Further, parallel programs are seldom portable: a program that executes effectively on one concurrent machine (for example, a Cray X-MP) cannot in general be assumed to work with comparable efficiency on another concurrent architecture (for instance, a hypercube), and it is difficult to make the necessary transformation.

Thus it is not only vital to develop tools to assist in all phases of parallel program development: the design of such tools must pay due regard to the crucial issues of performance and source code portability. So, whilst paying attention to such issues as the provision of a suitable user interface, we must judge a parallel programming environment primarily by its ability to provide:

- high target code efficiency
- portability
- powerful automatic support for program development, debugging, and maintenance.

In this paper, we claim that - as a result of the complexity of the task - these objectives can only be achieved by using a knowledge-based approach, supported by extensive analysis facilities, and with a special emphasis on performance analysis and prediction. The paper describes the design principles for a multi-layer, multi-target, intelligent programming environment for the efficient solution of data-parallel numerical algorithms on parallel machines. A key role in this environment is played by the new Fortran standard Fortran 90, which serves as a target (for transformations from higher language levels) as well as a source language.

*The work described in this paper is being carried out as part of the research project "Virtual Shared Memory for Multiprocessor Systems with Distributed Memory" funded by the Austrian Research Foundation (FWF) under the grant number P7576-TEC. The authors assume all responsibility for the contents.

The paper is structured as follows: In Section 2, we outline the capabilities of state-of-the-art program restructurers for parallel programming, and identify some important elements which are missing in the general approach taken. We then introduce the main features of an advanced parallel programming environment which overcomes these drawbacks, and take a closer look at the kinds of knowledge which will be incorporated in such a system. In Section 3, we describe the major design features of such an environment. The paper closes with a brief summary (Section 4).

2 Current Programming Environments

In this section, we outline the design of three existing program restructuring systems for numerical computation, which were developed to assist in the task of generating parallel code for several different architectures. They all operate on sequential FORTRAN programs.

2.1 The Design of Three Existing Systems

2.1.1 Parafrase

Parafrase is a tool for transforming a FORTRAN66 or FORTRAN77 program into a semantically equivalent extended Fortran program adapted to meet the requirements of a particular kind of machine. Developed at the University of Illinois at Urbana-Champaign in a long-term research effort centered around D. Kuck, it was the first systematic attempt to transform sequential Fortran programs into concurrent programs and was aimed at register-to-register and memory-to-memory vector machines, array processors, and shared-memory multiprocessors ([Kuck 84],[Poly 86]).

To restructure a program with Parafrase, users input a sequential program together with a list of the transformations they require. These transformations are encoded as procedures that may read and/or write the program; they are applied in the order specified in the list and each one performs a source-to-source transformation. Over 100 different transformations are available, ranging from architecture-independent ones to machine-specific transformations, which adapt the program to the requirements of a particular target machine. Parameters, known as switches, are used both to control such functions as printing or debugging and to provide transformations with information. The user may also convey information to Parafrase by means of assertions and commands embedded into the source program. Thus the user must specify every transformation performed on the input code, paying due regard to those which will result in code suitable for the target system architecture. Note that this is a laborious task requiring a great deal of expertise on the part of the user, and that the transformation process itself is slow, since each pass is implemented as a source-to-source transformation. On the other hand, Parafrase can be easily extended by new transformations.

To help the user evaluate the quality of the results of transformations, Parafrase provides statistics to show the fraction of loops in a program that have been restructured.

2.1.2 PFC and PFC$^+$

PFC (Parallel Fortran Converter) is an automatic source-to-source vectorizer that translates from Fortran 66 or 77 into Fortran 8X. It has been under development since 1979 by K.Kennedy and his group at Rice University, Houston, and was initially based on Parafrase ([AllKe 82]).

PFC first transforms the source program into an internal representation similar to those used in conventional compilers, and essentially based upon an abstract syntax tree and the associated symbol table. It carries out this task with the efficiency of a compiler for a sequential computer. In addition to standard data flow analysis, the tool performs data dependence analysis and applies a set of standard transformations. Interprocedural analysis is performed. All PFC transformations manipulate the internal representation, rather than making a source-to-source transformation, which makes the system more than an order of magnitude faster than Parafrase ([AllKe 87]). However, it provides much fewer transformations and there are fewer options available to the user.

PFC$^+$, an extension of PFC, implements Callahan's parallel code generation algorithm for shared memory systems ([ACK 87], [Call 87]). In addition, elements of an interprocedural dependence test have been implemented.

The efficiency of this kind of approach is encouraging: the extensive analysis it applies provides essential information for the task of restructuring: however, not all information required in this process can be derived statically.

2.1.3 SUPERB

SUPERB is an interactive restructuring tool which translates Fortran 77 programs into concurrent SUPRENUM Fortran programs for the SUPRENUM computer ([Giloi 88]), a distributed-memory multiprocessor whose computing nodes, in turn, contain pipelined vector units. Thus SUPERB restructures in two phases: First coarse-grain parallelism is determined and program execution is distributed over a set of processes: then it vectorizes the resulting code for the individual nodes of the machine. The general approach taken for restructuring in SUPERB is based on data partitioning:

SUPERB assumes a *Single-program-multiple-data (SPMD) model*, such that each process executes the same program, but is applied to different portions of the data domain. At the outset of parallelization, the user must specify a data partition for each array in the program. This is a crucial step, since the way in which the program's data is partitioned and mapped determines the process structure of the parallel program and in particular, the communication required; hence, it also determines the overall performance of the parallelized program. The subsequent creation of tasks and generation of communication is performed automatically.

The system's front end first transforms a Fortran 77 program into an intermediate representation consisting of an attributed abstract tree, an associated symbol table, flow graphs, and initial data flow information. The program representation is then normalized, making the subsequent application of transformations simpler and more efficient.

The core of the system comprises a set of routines in an analysis component and the transformation catalog; these routines manipulate the program's intermediate representation. The analysis component furnishes a collection of tools for program flow and dependence analysis, using both intraprocedural and interprocedural analysis techniques.

The interactive component acts as an interface to allow control of the other system services by establishing a two-way communication link between the user and the modules of the system. Menus are provided so that the user may select individual transformation strategies or request other services from SUPERB. At the end of a restructuring session, the back end uses the information contained in the transformed intermediate representation to produce a concurrent SUPRENUM Fortran program containing message passing operations and vector instructions. SUPERB puts a good deal of effort into optimizing the target program, in particular extracting communication from loops whenever possible, and combining individual communication statements (by vectorization and fusion) to reduce the overall communication cost ([GerZi 90]).

SUPERB was developed at Bonn University under the direction of H. Zima; a more detailed description of the system is to be found in [ZBG 88], [Gernd 89].

2.2 The Limitations of Current Programming Support Systems

There are a number of reasons why the approaches described above to build restructuring systems will not, of their own, suffice for the highly complex and challenging task of providing a mature programming environment for parallel systems. These systems will need to retain efficiency while providing a level of guidance and a degree of flexibility not achieved hitherto, and must satisfy the needs of a broad spectrum of users, some of whom will require an integrated system to work as automatically as possible, whereas others will expect tools to provide them with detailed and precise information to enable them to make the major strategic decisions for parallelization. We discuss some of the missing features more fully below (cf. also [ChaHe 91]).

2.2.1 Current Restructurers Lack Explicit Structured Knowledge

A substantial amount of knowledge was used to build the current generation of restructuring systems, and it has been incorporated into them. This knowledge is of several kinds, and includes facts about a given architecture or range of architectures, the target system's *software environment* and the related *programming paradigm*. A precise knowledge of preconditions for valid program transformations was used to enable the restructurer to test for these and perform the appropriate transformation automatically when instructed to do so. The knowledge of a few *useful sequences* of transformations enabled an automatic approach to certain tasks. But this knowledge is buried in many thousands of lines of code, where it cannot be easily accessed, modified or extended; nor can it be used to reason about the transformation process.

Some restructurers perform a good deal of advanced program analysis and store the results in data structures which provide an easily accessible base of information about the program. But apart from some low-level information (such as details of data dependence), this information is hardwired into the program, in a form which makes it inaccessible to the user.

Parallel computing systems are in a constant state of change: not only are new architectures regularly introduced, existing machines are modified extensively or extended to increase their overall performance, and compilers and system software are often updated. Current parallelizing tools have not been designed with this kind of change in mind: the implicit nature of the knowledge they embody makes them *inflexible*, and in general, extension or modification will involve a major programming effort.

2.2.2 Current Systems Lack "Intelligence"

Apart from certain pre-defined sequences of transformations, which often have the task of normalizing programs, current systems require the user to specify which transformations are to be applied to the program and (where relevant) how and where to apply them. This requires almost complete manual control of the overall transformation process.

These restructurers offer no *guidance*: they do not help the user to select certain regions of the program for transformation or further analysis, nor do they suggest a certain transformation or set of transformations for application. Further, they generally lack adequate facilities for evaluating the effect of transformations once they have been applied. The crucial decisions are the responsibility of the user. But many of the decisions to be taken cannot be intuitively answered, even by an experienced programmer of parallel systems. There are different, and sometimes conflicting, goals to be attained (e.g., load balancing vs. minimization of communication), and non-trivial trade-offs to be considered.

Further, restructurers are currently not able to *advise* the user about ways to improve his program: restructuring systems are programmed in such a way that they will recognize and transform certain kinds of codes better than others. But the only type of information current systems can provide the user with about his program is at a very low level and consists of such objects as symbol tables, call graphs and dependence relationships between statements.

Thus present-day restructurers do not achieve some of the major goals for advanced parallel programming tools: *they do not relieve the user from the burden of* understanding fine details of the target machine, its programming paradigm and the process of program restructuring itself. The quality of the target code depends largely on the expertise of the user in all of these areas.

2.3 What Kind of Support is Required?

If we want to construct tools which facilitate the process of parallel program development and produce efficient code, it is essential that we overcome these deficiencies. We therefore expect that future programming support systems should:

Contain explicit knowledge about:

- architectures, machines, languages, compilers, and restructurers
- the program: how it performs (algorithmic properties) and what it does (functional specification)
- problems: standard numerical problems and their solutions, i.e. their efficient implementation on a given set of architectures
- strategies: what goals to pursue and how to do so

Be interactive: Even automatic parallelizing tools cannot perform without assistance from the user. One of the main reasons for this is the undecidability or intractability of many relevant problems and the lack of adequate heuristics for handling them; furthermore, a static analyzer will have no information at all on variables whose value is input during program execution. The user will play an important role, informing the system of global relationships (some of which may be due to high-level properties of the algorithm) which an automatic state-of-the-art tool cannot detect.

Be easily modifiable: It will be necessary to retarget existing systems, to extend them by new transformations and to modify them in response to target system developments. Since in particular architectural and software system changes are rapid, it must be possible to adapt to them without a major reprogramming effort. This may be greatly facilitated by the explicit representation of knowledge within the system.

Have tools for performance prediction and measurement: Performance tools will not only identify the most important areas of existing code. They will also support the selection of appropriate algorithms, and help determine appropriate transformation strategies. This may involve invoking a performance prediction tool, running selected areas of the corresponding code with sample input values, or accessing performance information stored within the knowledge base.

Since the primary motivation for using parallel systems is to attain high performance, the environment must provide appropriate tools for measuring and analyzing achieved performance; feedback from these tools is of critical importance for a knowledge-based system.

2.4 Elements of a Knowledge-Based Program Development System

An advanced programming environment should be organized as a collection of *knowledge-based subsystems* with different levels of expertise. The advantages of this approach are threefold: 1) The possibility of rapid prototyping, 2) relatively easy modification of the knowledge-based system if the underlying system changes, and 3) the availability of an explanation facility. So far, few attempts have been made to develop restructuring systems using these techniques ([Bose 88a],[Bose 88b],[WanGa 89],[Wang90]).

The different kinds of information available should be organized into knowledge bases containing information about the application program, target environments, rules for transforming programs and facts about performance. It must be interactively accessible by the user and all tools in the system.

Objects in the knowledge base pertaining to an application program may include units (procedures, declaration libraries, macros, etc.), modules (structured collections of units), and programs. Objects are stored in an internal representation (procedures, for example, may be represented as abstract trees, together with a symbol table). They may exist in more than one version: for instance, a program may be present in its original form and in one or more transformed versions.

Information about such objects may include: the control flow graph, the results of data flow analysis, the call graph, the dependence graph, performance information, and linkage information.

The set of tools in a development system should be *integrated*: there should be a well-defined interface not only between the tools and the user, but also between these and the information stored in the system. Explanation facilities should guide the user in the complex process of program restructuring.

Knowledge relating to transformations should be organized in a modular fashion.

Within the environment, different goals may be pursued by different parts of the system, thus requiring specific strategies for applying sets of transformations. The following section (Section 3.1), for example, describes a system with several distinct layers, each of which performs a different kind of program translation. Goals may be achieved by pursuing sub-goals, which also require knowledge on how transformations may be used to do so. Thus this information will be hierarchical. It includes:

General Transformations This kind of knowledge relates to the languages involved and to the preconditions and effects of general transformations, such as those for normalization and standardization, loop interchange, scalar expansion, etc. For example, the effect of loop interchange can be described by specifying the modification of array access patterns.

Architecture-Specific Transformations Some transformations relate specifically to a certain architecture (such as SIMD, shared-memory MIMD, or distributed-memory MIMD), without exploiting specific machine features.

Machine-Specific Transformations Knowledge to generate code for a particular target machine involves identifying and applying a specific set of machine-specific transformations which address the detailed properties of a particular machine, such as registers, local memories, and the communication mechanisms. The system should be structured in a such a way that it is easy to include knowledge about new machines.

The sources of *knowledge acquisition* are primarily [ZiCh 90]:

- Papers and books about the subject,
- the properties of existing restructuring tools,
- human experts in the field, and
- the results of experimentation with various systems and characteristic numerical programs.

Note: A more advanced approach than the one we have described so far would attempt to "understand" the essence of an algorithm on a very high level and perform a transformation into a possibly completely different parallel algorithm at that level. While a general solution of this problem does not exist, we expect that a number of standard patterns can be identified for which automatic matching can be performed.

3 The Design of an Advanced Programming Environment

3.1 System Structure

Figure 1 shows the structure of an advanced parallel programming environment designed at the University of Vienna. In this system, we strive to make the program development process as automatic as possible, implementing an expert system approach that is supported by extensive analysis and guided by knowledge about algorithms, architectures, program performance and heuristics. On the other hand, since automatic translation will not produce optimal or near-optimal results under all circumstances, and will require user support for some tasks, the system will be designed to be interactive, allowing the user to furnish relevant strategic information and the system to explain the reasoning behind its transformation decisions.

It contains five levels, each of which is associated with one or more languages, starting with the highest, problem-oriented level (level 1), down to the most concrete, target machine specific level (level 5):

- Level 1: The Problem-Oriented Specification

  This level provides a set of problem-oriented languages which allow the user to specify a problem in the terminology of the application domain. These are restricted to numerical problem domains. For example, the equilibrium states of physical systems may be described by elliptic equations; problems in this field generally require the determination of a function that satisfies a given partial differential equation (PDE) on some domain, and some additional conditions on its boundary.

  Level 1 enables a problem formulation only; no reasoning about algorithmic solution approaches is done at this level.

- **Level 2: The High-Level Algorithmic Language**

  HAIL (High-level Algorithmic Imperative Language) is an object-oriented, high-level algorithmic language which allows the user to specify algorithmic solutions for numerical problems at a very high level of abstraction. In particular, a flexible abstract data type specification facility permits the declaration of problem-oriented data structures such as grids, grid hierarchies, trees, vectors, and matrices together with the associated operations. Furthermore, abstract control mechanisms allow the formulation of iterations over partially ordered sets in a way which is architecture-independent. Some important concepts in HAIL are derived from the language SUSPENSE ([RuWi 89]), developed at Bonn University as part of the SUPRENUM project.

This level serves two purposes: First, if the user has specified the problem at level 1, then its realization at the algorithmic level requires the selection of a solution method and its expression in HAIL. This transformation will be supported by an **expert composer**, which uses an approach based on knowledge about the domain of discourse, the high-level algorithmic languages, the formulation of algorithms at that level, and the target architecture.

Alternatively, the user may directly enter level 2 by explicitly formulating the problem solution in HAIL, bypassing the language features provided at level 1 and the expert system support for their transformation.

It is our objective to design HAIL in such a way that it can serve as the principal basis for program development, testing, and maintenance. This means that, in general, the program development process from level 2 downward should be completely automatic, and that, in particular, the user should only have to deal with code from levels below that of Fortran 90 if the tools have failed to generate target code with an adequate measured performance (but can access and modify code at all levels if desired).

- **Level 3: The Algorithmic Language (Fortran 90)**

The language of level 3 is the new Fortran standard Fortran 90 (F90), extended by an annotation language (AL). AL allows the formulation of assertions that can be used to guide the translation of F90 programs into lower-level code and to enhance the efficiency of the translation process as well as the quality of the generated target code. Assertions may be related to knowledge existing at a higher level or may be explicitly provided by the user. They can be either global or bound to specific program units.

The principal reason for the pivotal role played by F90 is our expectation that F90 will be accepted and adopted by the Fortran community as the new standard for programming numerical problems. As a consequence, F90 will serve as the main programming language in this field, and at the same time establish a generally acknowledged, machine-independent language level which provides portability across a wide range of architectures. These advantages far outweigh the difficulties involved in suitably defining AL and guaranteeing the proper transfer of high-level information to lower levels.

Thus a problem solution described with the means available at level 1 or 2 is translated into a portable code version at level 3, which may be transferred to other systems or users. We permit, however, a direct translation from a level 2 program specification to the levels 4 and 5, in order to obtain maximum efficiency for the compilation process as well as the generated target code.

- **Level 4: Enhanced Algorithmic Language (FORUM)**

FORUM is a F90 superset which contains language features for the specification and control of parallelism in an architecture-independent way. We plan to define a language that can be efficiently translated for a variety of architectures, including at least distributed-memory systems (DMS) and hierarchically structured shared-memory systems.

Examples of FORUM features (in the context of DMS) are specification statements for the distribution and alignment of arrays (cf. [CHZ 91], [BCSZ 91]).

In view of the low level of FORUM, we do not expect that users will routinely formulate programs in this language. However, situations will arise in which a tool such as an automatic data partitioner does not yield acceptable results. In these circumstances, the user must have the option of interactively controlling the translation process, thus directly influencing critical implementation decisions.

As already mentioned, we expect FORUM to be the target of a translation from either level 2 or 3.

- **Level 5: Target-Oriented Fortran Dialects (DIALs)**

Level 5, the lowest level considered in our programming environment, is the level of target-machine oriented Fortran dialects. This includes languages such as SUPRENUM Fortran ([Comp 89]) for the SUPRENUM supercomputer, CM Fortran for the Connection Machine CM-2, and Cedar Fortran, and will include machines with any of the architectures mentioned above. Note that level 5 consists of source languages only, and single-node compilation is not considered here.

3.2 Levels and Languages

In this section we present some general features of levels as introduced above, and examine the relationships between different levels and the associated languages.

Let S denote a language at level i $(1 \leq i \leq 5)$. Then S will satisfy certain properties and a number of facilities are available in conjunction with the language, independent of the specific choice:

- Associated with S there is a **semantic model** which forms the basis for reasoning about programs at level i.
- A **knowledge base** KB(S) stores knowledge about the semantics of S, including, for level $i > 1$, knowledge about the efficient formulation of algorithms in S. KB(S) also contains information relevant to intelligent guidance of the application of program transformations (depending on criteria such as performance) and reiated heuristics.
- **Analysis tools** perform a static (i.e., compile-time) examination of program properties and carry out certain standard transformations. These include:
  - constant propagation
  - the determination of definition-use and use-definition chains
  - dependence analysis
  - performance analysis (in dependence on problem size and architecture)
- An **editor** is provided which offers the user a means to formulate programs in S or change existing programs interactively. In addition to syntactic knowledge, the editor has access to the knowledge base KB(S) and communicates with the user via an assertion and command language that is closely tied to the semantic model associated with S. Furthermore, the editor is supported by a set of incremental analysis tools.
- An **interpreter** for S provides a rapid prototyping facility.
- A **debugger** for S allows the user to test and debug programs at the level of S.
- A set of **restructuring transformations (RTs)**, which preserve the semantics of programs. The main purpose of RTs is the modification of the structure of programs in such a way that the match between the parallelism recognized at a given level and the parallelism at the next lower level is improved. For example, the application of loop distribution to a multi-statement Fortran loop creates a sequence of single-statement loops, which may be translated to vector statements at a lower level.

In addition to the RTs that are associated with a given language S, there is a set of **inter-level transformations (ILTs)** for any pair of languages S,S', where S is of level $i \leq 4$ and S' is of level $i+1$. ILTs translate – in a semantics-preserving way – elements of S to elements of S'. These transformations implement the translation from more abstract to more concrete programs by a process of step-wise refinement.

The whole process of translating a specification P1 (at level 1) into a program P5 formulated in a target-oriented Fortran dialect is performed by applying a sequence of RTs and ILTs to P1. For example, if P1 is a PDE that is defined on some domain, with an additional set of boundary conditions, then the major steps of transforming P1 into a program P5 in INTEL iPSC Fortran may be as follows (Pi denotes a program at level i):

P1 – Problem specification
PDE on domain R with boundary conditions

The application of the **expert composer** to P1 implies the discretization of R and the selection of a solution method (e.g., the multigrid method with a corresponding choice of a relaxation method and a global control algorithm such as V-cycle or W-cycle), which yields a HAIL program, P2:

P2 – High-level language program

P2 specifies an algorithmic solution to the given problem in the framework of an object-oriented approach which allows the use of domain-specific data types (such as grid hierarchies) in combination with abstract, high-level control structures (such as iterators over grid hierarchies).

The high-level constructs of P2 can be translated step-by-step into the constructs of the procedural programming language F90. For example, a HAIL-iterator applying a given operation to all elements of a grid according to some partial order specified by an order star may be translated into a F90 DO-loop, annotated by an assertion stating the original partial order.

P3 – F90 program

P3 specifies a solution of the problem at a machine-independent, portable, intermediate procedural level. At this level, special library functions may be included in the program.

The only explicit parallelism contained in P3 is the fine-grain parallelism of array operations and statements. The translation to the next-lower level (FORUM) provides the basis for making the coarse-grain parallelism in the program explicit. This can be done by generating statements for the partitioning and alignment of arrays, based on an extensive analysis process of the program's reference patterns:

P4 – FORUM program

P4 specifies a solution of the problem at a low procedural level.

The transformation from P4 to P5 is performed by modifying P4 according to the data partition specified in P4. This process consists of the three steps masking, generation of communication, and optimization, which are similar to the corresponding actions in SUPERB. The optimization step is crucial for obtaining efficiency at run-time as it recognises certain standard communication patterns and transforms them into efficient collective communication routines. This produces the final version:

P5 - INTEL iPSC Fortran program

The compilation of P5 is not directly of concern to us; however the generation of P5 has to take into account the capabilities and idiosyncrasies of the compiler as well as the target machine, as discussed above.

4 Conclusion

In this paper, we introduced the major features of an advanced programming environment, and described the structure of an intelligent hierarchical programming environment to support the development of data-parallel numerical programs. This system makes extensive use of knowledge-based and performance analysis techniques. Other research in this area has been pursued, in particular, by Wang at Purdue University ([Wang90], [Wang85],[WanGa 89]).

References

[BCSZ 91] P. Brezany, B. Chapman, A. Schwald, and H. Zima: Vienna Fortran: A Fortran Extension for Distributed-Memory Systems. Technical Report, Austrian Center for Parallel Computation (to appear)

[ACK 87] Allen J.R.,Callahan D. and Kennedy K: Automatic Decomposition of Scientific Programs for Parallel Execution Conf. Rec. Fourteenth ACM Symp. Principles of Programming Languages (POPL), 63-76, 1987

[AllKe 82] Allen J.R. and Kennedy K.: PFC: A Program to Convert Fortran to Parallel Form, Proc. IBM Conf. Parallel Comp. and Scientific Computations, 1982

[AllKe 87] Allen J.R. and Kennedy K. (1987): Automatic Translation of FORTRAN Programs to Vector Form ACM TOPLAS 9,491-542, 1987

[Bose 88a] Pradup Bose: Heuristic Rule-Based Program Transformations for Enhanced Vectorization. Proc. Int. Conf. on Parallel Processing, 1988

[Bose 88b] Pradup Bose: Interactive Program Improvement Via EAVE: An Expert Adviser for Vectorization. Proc. International Conference on Supercomputing, St. Malo, July 1988

[Call 87] Callahan,D.: A Global Approach to Detection of Parallelism. Ph.D.Dissertation, Department of Computer Science, Rice University, Houston, Texas, 1987

[ChaHe 91] Chapman B. and Herbeck H.: Knowledge-Based Parallelization for Distributed Memory Systems. Proc. First Int. Conf. of ACPC, Springer Verlag (to appear). Also: Technical Report Series ACPC/TR 91-11, Austrian Center for Parallel Computation (1991)

[CHZ 91] Chapman B., Herbeck H. and Zima.H.: Automatic Support for Data Distribution. Proc. DMCC6, Portland, OR IEEE Computer Society (to appear)

[Comp 89] Compass, Inc.: SUPRENUM Fortran Reference Manual. Compass, Inc., Wakefield MA (August 1989)

[Gernd 89] H.M. Gerndt: Automatic Parallelization for Distributed-Memory Multiprocessing Systems. Ph.D. Dissertation, University of Bonn, Technical Report Series ACPC/TR90-1, Austrian Center for Parallel Computation

[GerZi 90] Gerndt,H.M., Zima,H.: Optimizing Communication in SUPERB. Proc. CONPAR 90-VAPP IV (Zurich), LNCS 457, 300-311. Also: Technical Report Series ACPC/TR 90-3, Austrian Center for Parallel Computation (1990)

[Giloi 88] Giloi W.K.: SUPRENUM: A Trendsetter in Modern Supercomputer Development, Parallel Computing 7 (1988), 283-296

[Kuck 84] Kuck D.J., Kuhn R.H., Leasure B. and Wolfe M.: The Structure of an Advanced Retargetable Vectorizer, In: [Hwang 84 a], 967-974, 1984

[Poly 86] Polychronopoulos C.D.: On Program Restructuring, Scheduling, and Communication for Parallel Processor Systems, Ph.D.Dissertation, Tech. Report SCRD No.595, Center for Supercomputing Research and Development, University of Illinois at Urbana-Champaign, Aug 1986

[RuWi 89] Ruppelt T. and Wirtz G.: Automatic Translation of High-Level Object-Oriented Specifications into Parallel Programs. Parallel Computing 2 (1988)

[WanGa 89] Wang K.-Y.,Gannon D.:Applying AI Techniques to Program Optimization for Parallel Computers. In (Hwang K.,DeGroot D. eds.): Parallel Processing for Supercomputers and Artificial Intelligence ,12,441-485, McGraw-Hill Publishing Company (1989)

[Wang85] K. Wang: An Experiment in Parallel Programming Environment: The Expert Systems Approach

In: K.S. Fu (Ed.): Some Prototype Examples for Expert Systems, TR-EE-85-1, Electronic Engineering School, Purdue University, 1985, 591-624

[Wang90] K. Wang: A Framework for Intelligent Parallel Compilers
Tech. Report CSD-TR-1044, CER-90-52, Dept. Computer Science, Purdue University, November, 1990

[ZBG 88] Zima H.P., Bast H.-J. and Gerndt H.M.: SUPERB - a tool for semi-automatic MIMD/SIMD parallelization. Parallel Computing, 6, 1-18 (1988)

[ZiCh 90] Zima,H.P., Chapman,B.M.: Supercompilers for Parallel and
Vector Computers, ACM Press Frontier Series/Addison-Wesley (1990)

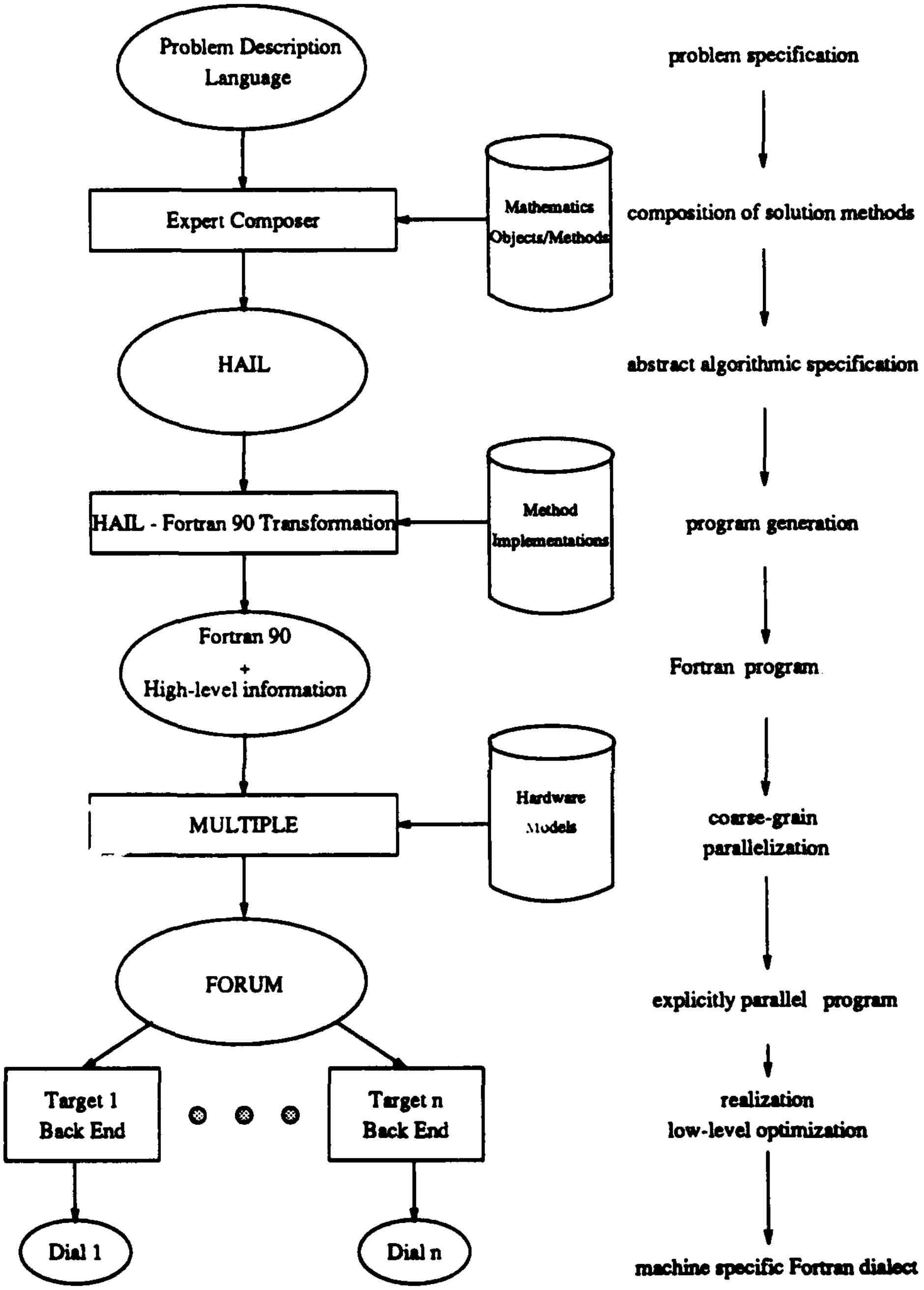

Problem Description Language
Expert Composer
Mathematics Objects/Methods
HAIL
HAIL - Fortran 90 Transformation
Method Implementations
Fortran 90 + High-level information
MULTIPLE
Hardware Models
FORUM
Target 1 Back End
Target n Back End
Dial 1
Dial n
problem specification
composition of solution methods
abstract algorithmic specification
program generation
Fortran program
coarse-grain parallelization
explicitly parallel program
realization low-level optimization
machine specific Fortran dialect

Figure 1: The structure of an advanced programming environment

Fortran Interface Blocks as an Interface Description Language for Remote Procedure Call

Paul E. Buis
Wayne R. Dyksen
John T. Korb

Abstract

In this paper, we discuss the use of Fortran 90 interface blocks as an interface description language (IDL) for remote procedure call (RPC). An implementation of a stub generator based on this language is used to generate an interface between Fortran 77 and an innovative RPC system. An example application is remote access to scientific subroutine libraries such as IMSL.

1 *Introduction*

With the existence of large networks of heterogeneous computers comes the desire for truly distributed scientific computing. Users want to formulate their problems on one machine, solve them on a second and visualize the results on yet a third machine, all within one problem solving environment. They want to apply transparently the power and special architectures of the machines on their networks to the appropriate parts of their problems without learning the intricacies of new operating systems or compilers. Users want to exploit this network power transparently, from their desk, without remotely logging into another system.

To this end, we are exploring the application of distributed systems technologies to scientific computing. We have developed a new paradigm for client-server interaction called *remote interpretation* in which the server executes a high-level language interpreter and the client sends expressions in this language to be evaluated. Remote interpretation can be used to simulate more traditional forms of client-server interaction, such as remote procedure call (RPC). In order to construct large distributed systems, we are building an automated programming tool to write the message passing code for client-server interaction. The input to this programming tool, called a stub generator, consists of Fortran 90 interface blocks. This paper describes the use of Fortran 90 interface blocks for this purpose.

2 Remote Interpretation

In this section, the idea of using an interpreted programming language as a control protocol for client-server communication is explained. The NeWS window system uses the PostScript language as an input language for its server. PostScript is a general-purpose stack-oriented language with good image processing primitives. In the NeWS system, clients send PostScript expressions to the NeWS server where they are evaluated. This section discusses the use of the Scheme programming language [3] in an analogous way. Scheme is a general-purpose language with both good numeric and symbolic processing primitives making it appropriate for use in scientific systems.

For the purposes of discussion, the use of an interpreted programming language as a control protocol will be called *remote interpretation.* This section discusses an implementation of remote interpretation using the Scheme language, and Sun XDR, concluding with a description of how to emulate RPC with remote interpretation.

2.1 NetScheme

Scheme is a small Lisp-like language. It shares the syntax of Lisp and uses a subset of Lisp operators. The primary difference is that names are statically scoped rather than dynamically scoped. Scheme has a rich set of numeric types: integer, rational, real, and complex. The inclusion of real and complex numeric types is essential for use in scientific computing. Scheme also supports lists and vectors.

NetScheme is our implementation of a Scheme subset with extensions for distributed scientific computing. The NetScheme interpreter uses TCP sockets for all input and output. The use of TCP enables both portability and large geographic separation of clients and servers with reasonable reliability.

2.2 XDR

NetScheme uses the Sun XDR protocol for its data representation protocol. That is, rather than sending printable ASCII strings to communicate data values, a binary representation is used that conforms to the Sun XDR data representation standard.

NetScheme is interfaced with the XDR protocol via four functions that are built into the NetScheme interpreter. `xdr-bind` is used to start up the XDR connection between client and server. `xdr-recv` is used to receive a vector of homogeneous elements from the client. `xdr-send` is used to send a vector of homogeneous elements back to the client. `xdr-flush` is used to flush the output buffer of the server to the client.

The programming interface to XDR is specialized to the sending of arrays of data rather than individual elements. The specialized code has been observed to reduce the number of user-level CPU cycles by a factor of 10 and increase throughput by a factor of 2.

2.3 Simulating RPC with NetScheme

NetScheme can be used by a client to simulate remote procedure call. Recall that in a remote procedure call system, the client sends a message containing input parameters to the server, the server carries out a computation on the behalf of the client, and the server sends the results to the client in a reply message. To simulate this, a client can cause the server to execute an `xdr-recv`, a function evaluation, and then an `xdr-send` to achieve the same effect. For example, if the client wants to remotely call a function `f`, it can emit "`(xdr-send (f (xdr-recv))) (xdr-flush)`" to the server and then execute the code to send and receive the data over the XDR connection.

The advantage of using this RPC simulation is in its transparency to the client designer. The designer can be blissfully ignorant of the fact that some of the computations are being done remotely. The client implementor is responsible for packaging the RPC simulation into a single procedure, hiding the fact that the computation is not local. The next section describes tools that will relieve the client implementor of the burden of writing this code, making client construction very easy.

```
Interface
Subroutine LSARG(N, A, LDA, B, IPATH, X)
Integer N, LDA, IPATH
Real A, B, X
Intent(In) N, A, LDA, IPATH
Intent(Out) X
Dimension A(LDA,N), B(N), X(N)
End Subroutine
End Interface
```

Figure 1: Example Fortran 90 interface block

3 An Interface Description Language

The interface description language (IDL) for an RPC system often parallels the declaration constructs from the programming language used by its intended clients. We wanted an IDL that matched well with Fortran, since that is the primary language currently in use in scientific computing. To match well, an IDL should have the ability to describe *any* subroutine interface possible in Fortran. While Fortran 77 lacks pointers and aggregate types, it does have *conformant* multidimensional arrays; i.e., arrays with dimensions specified by the value of other arguments at call time. Another powerful feature of Fortran argument passing is procedural parameters; i.e., passing one subprogram as an argument to another subprogram. While these features are directly supported in Fortran, they are not directly supported by any of the existing IDLs.

Fortran 90 [1], a proper superset of Fortran 77, contains a syntax for an *interface block*. The Fortran 90 interface block syntax is an almost perfect match for an interface description language for distributed systems. For each subprogram described in an interface block, all of the declarations are included that specify the order, names, and types of each argument. These declarations use the exact same syntax as is found in the executable code for the subprogram. In addition to the declarations found in Fortran 77, Fortran 90 adds an INTENT statement declaring an argument to be for input only, output only, or for both input and output. An example interface block is given in Figure 1.

From the information provided in such an interface block, one can generate the structure of the request and reply messages needed in an RPC system. The request message consists of those arguments with an input intent, and the reply message consists of those arguments with an output intent. The message consists of the value of each argument in the order the arguments are specified. The value of array arguments includes not only the values of the elements of the array, but also the total number of elements in the array.

We have constructed an interface description language [2] using a subset of the syntax for Fortran 90 interface blocks that allows specification of virtually all Fortran 77 subprograms. We only needed to make two small modifications to the Fortran 90 standard. First, while the Fortran 90 standard permits arguments to have unspecified intent, we require that all arguments used for input or output must have their intent declared. Fortran compilers simply assume that unspecified intent is equivalent to an intent of *both* input and output, and will therefore avoid making certain optimizations. In our language, we assume that unspecified intent is equivalent to an intent of *neither* input nor output; i.e., a scratch argument used only for the storage space it provides the callee. This deviation from the Fortran standard allows us to optimize the message passing to avoid unneeded transmission of scratch values. Second, while the Fortran 90 standard permits an array declaration to not specify the size of the trailing dimension, we require that the size of all the dimensions of an array be specified. To compensate for this additional strictness, we allow a broader class of size specifications than the Fortran 90 standard which allows only literal integer constants or simple integer arguments to specify a size. We expand the size specification to allow for arithmetic expressions involving addition, multiplication, division, modulo, maximum, and minimum operators, all using standard Fortran syntax.

Since arguments are passed by value in both directions in such a system, a potential for a subtle change in semantics exists if the target language does not use pass-by-value-result semantics. Fortran is typically thought of as pass-by-reference, but the Fortran standards have made the unenforced stipulation that *aliasing* (i.e., having more than one argument contain the same location in memory), is illegal. Because of the prohibition of aliasing, no changes in the argument passing semantics occur when a pass-by-value result mechanism is used in implementation. Hence, an RPC system for Fortran can avoid any changes in argument passing semantics.

4 Stub Generation

We have constructed a stub generator based on this IDL that generates stubs for NetScheme. On the client side, RPC is simulated as outlined in Section 2.3. On the server side, code is generated to add new built-in functions to the NetScheme interpreter. Hence, the stub generator is used to construct servers, even if the client will not perform RPC simulation.

The server stub generator provides the code to add Fortran routines into the NetScheme interpreter. The NetScheme interpreter requires each built-in function be implemented in C (since the NetScheme interpreter is written in C) as a function with an argument list as its input parameter and returning a vector of output values.

The stub generator is also capable of producing client stubs that simulate remote procedure call. The use of such stubs is optional in the NetScheme system. The automatically generated client stubs use both the TCP link for the control protocol and the separate TCP link for the data representation protocol. Some care must be taken that the two links are properly synchronized to prevent deadlock.

As a first step, the client stub uses the TCP link for the control protocol by calling **fprintf** repeatedly, once for each Scheme expression to be evaluated by the server. First, a sequence of Scheme expressions is output to define input parameters by invoking **xdr-recv**. Second, an expression is output to invoke the server stub. Third, a sequence of expressions is output that invoke **xdr-send** to instruct the server to send each output argument to the client over the data representation connection. Fourth, a call to **fflush** is executed on the output buffer associated with the TCP link.

As a second step, the client stub uses the TCP link for the data representation protocol to send each input argument to the server. As the server executes **xdr-recv**, it inputs an array of data. After all input arguments have been output by the server, it calls **xdr_flush** to be sure that all buffered data is actually sent to the server. Thus, if the total size of the input arguments is sufficiently small, they all may be sent in a single TCP packet.

As a third step, the client stub uses the TCP link for the data representation protocol to receive each output argument from the server. Since both the TCP connection and the XDR routines are bidirectional, the same XDR routines are used in this step as in the previous one (with the exception of **xdr_flush**). At this point, the client and the server will be synchronized completely and the server will be idle.

5 Application: Remote Subroutine Libraries

This section explores an application of the approach described in the previous sections. Scientific programs often can be decomposed into a component written by an application programmer and components supplied in the form of subroutine libraries. The component written by the application programmer can reside in a client process and the components supplied as subroutine libraries can be packaged into server processes.

The construction of a remote subroutine library can be automated using the stub generators presented in the previous sections. First, one writes an interface block that describes all the public subprograms in the library. Second, the server stub generator is run to create a NetScheme server for the library. Third, the client stub generator is run to create a set of client stubs, one for each public subprogram. Finally, these stubs are placed in a stub library that takes the place of the original library when the client is linked.

5.1 Case Study: Remote IMSL

To demonstrate the feasibility of constructing remote subroutine libraries, we have constructed one for the IMSL Math/Library. We constructed interface blocks for 268 of the single precision routines. This process was partially automated by a Fortran compiler that produced interface blocks as an aid to optimization. However, the compiler was unable to determine the exact intent of most arguments and did not specify the trailing dimension of arrays. The total size of the resulting interface blocks is about 7000 lines.

The IMSL interface blocks were passed through the stub generators described in the previous sections. The resulting server stubs are about 17,000 lines in a single file and the resulting client stubs about 37,000 lines in 268 separate files. Clearly, the use of a stub generator is beneficial, since hand-coding and maintanence of such a large piece of code would be very difficult.

5.2 Performance Data

To demonstrate that the use of distributed systems is both practical and has potential performance benefits for scientific systems, we used an early prototype of our system to use a direct method for solving sparse, banded linear systems of equations of the sort that arise from applying finite difference methods to discretize three dimensional, partial differential equations

		Problem Size			
		8	10	12	14
Client	Server	Number of Equations			
Machine	Machine	256	1000	1728	2744
Sun 3/50	(local)	48.4	276.0	1028.0	—
Sun 3/50	Alliant FX/80	2.7	8.2	26.7	82.4
Sun 3/50	Cray XMP	2.0	3.2	5.4	26.0
Sun 4/110	(local)	6.9	32.3	171.0	570.0
Sun 4/110	Alliant FX/80	2.4	8.3	26.1	80.4
Sun 4/110	Cray XMP	2.0	3.2	5.4	26.0
Alliant FX/80	(local)	1.8	7.3	25.0	80.0
Cray XMP	(local)	0.1	0.4	1.2	19.6

Table 1: Elapsed Times in seconds to solve an elliptic PDE on various machines with and without using RPC

(PDEs). Although the same code is capable of solving more complex PDEs in the same amount of time, the particular PDE we chose as an example for timings was

$$\begin{aligned} u_{xx} + u_{yy} + u_{zz} &= f(x,y,z) \text{ on } \Omega \\ u(x,y,z) &= g(x,y,z) \text{ on } \partial\Omega \\ \Omega &= [0,1] \times [0,1]. \end{aligned}$$

The client runs on a small workstation (such as a Sun 3 or Sun 4) and forms the linear system of equations in ELLPACK standard format [4]. The system is solved by an RPC request to a vector computer (such as an Alliant FX/80 or Cray XMP).

The problem size, n, is the number of interior mesh points on each edge of the domain. The discretization results in a sparse system of n^3 linear equations with 7 nonzero coefficients per equation. The amount of communication necessary using ELLPACK sparse matrix format is $\mathcal{O}(64n^3)$ bytes, the amount of memory needed for 64-bit LINPACK banded storage is $\mathcal{O}(24n^5)$ bytes, and the amount of work is $\mathcal{O}(2n^7)$ floating point operations. So, for a problem size of $n = 10$, there are 1000 equations with about 64 kbytes to transmit, 2.4 Mbytes to store, and 20 MFLOPS to perform.

Table 1 summarizes the timing results. The Alliant was located on the same ethernet as the Suns, but the Cray was located off-site and accessed by a T1 line. One can see a distinct advantage in using RPC with a nearby fast server as opposed to solving the problem locally. While use of a more distant and faster server has some advantage for large problems, the raw data indicate a high variability in response times due to highly varying loads on both the remote machine and the network. Note that for machines on the local network, communication times are small compared to computation times.

6 Summary

We have shown that Fortran 90 interface blocks can serve as the basis for an interface description language for a distributed system. We have used such an interface description language to automatically construct stubs for the NetScheme system. The server stubs provide new built-in functions for the NetScheme interpreter. The client stubs simulate remote procedure call by sending an appropriate sequence of NetScheme expressions and following them up with an appropriate sequence of data transfers.

References

[1] American national standard for information systems programming language Fortran, March 1989. Draft S8, Version 111.

[2] Paul Buis, Wayne Dyksen, and John T. Korb. Fortran interface blocks as an interface description language for remote procedure call. Technical report, Computer Science Department, Purdue University, January 1990. CER-89-9/CSD-TR-953.

[3] Jonathan Rees and William Clinger. Revised^3 report on the algorithmic language scheme. *SIGPLAN Notices*, 21(12):37–79, December 1986.

[4] John R. Rice and Ronald F. Boisvert. *Solving Elliptic Problems using ELLPACK*. Springer-Verlag, New York, Berlin, Heidelberg and Tokyo, 1984.

An Integrated Problem Solving Environment for Numerical Simulation of Engineering Problems

Nobutoshi Sagawa, Donal P. Finn and Neil J. Hurley
Hitachi Dublin Laboratory, Trinity College, Dublin 2, Ireland.

ABSTRACT

This paper presents an AI based problem solving environment for numerical simulation of problems described by partial differential equations (PDEs). The system aims to allow users to carry out simulation without knowledge of mathematics or numerical analysis. Three abstract models (physical, mathematical and numerical models) are extracted as the essentially important steps to automatise the numerical simulation process. Transformation between these models is realised using two tailored knowledge-bases, which are organised in a frame based manner to reflect the hierarchical nature of the domain knowledge. Localised rule bases are bound to each frame definition to create the next level of the abstract model. This system has been coupled with an existing PDE solver and realises an integrated problem solving environment for engineering problems.

1 Introduction

With the development of powerful computers during the 1980s, many high-level numerical software packages such as PDEQSOL [Umetani et al 1985, Kon'no et al 1986], ELLPACK [Rice 1985] and FIDISOL [Schonauer and Schnepf 1987] became available for solving problems described by PDEs. Although such high-level software greatly reduces the effort demanded for numerical simulation compared with coding in FORTRAN or C, it is still necessary for users to supply considerable knowledge in order to carry out successful simulation. When the recent wide-spread use of workstations is considered, it can be regarded as an urgent issue to eliminate the need for specialised knowledge in order to allow users to exploit such high-level software.

A practical solution to this problem is to develop an AI-based problem solving environment that may be integrated with existing numerical software packages. On this basis, several research projects have been investigating the application of AI techniques to the area of numerical simulation. The Numerical Algorithms Group (NAG) [Chelsom et al 1990] are developing knowledge-assisted numerical routine selection tools for the diverse NAG FORTRAN library. The EVE system is aimed at mathematicians [Barras et al 1990] and enables the users to create PDEs from pre-defined primitive mathematical components. Another system which makes use of PDEs as the interface level was reported by [Russo et al 1987], in which some of the numerical stability and efficiency constraints are taken into account. These projects mainly aim at users who are knowledgeable enough to make the right choices during the decision process or who are to a certain extent familiar with mathematical expression.

This paper presents an AI based problem solving environment AI-PDEQSOL for numerical simulation described by PDEs. Unlike the problem solving environments discussed above, AI-PDEQSOL allows users to deal with practical engineering problems without requiring specialised knowledge of numerical or mathematical problem representation. Instead, the input to AI-PDEQSOL simply consists of engineering keywords and problem parameters. The output of AI-PDEQSOL can be dealt with by an existing PDE solver PDEQSOL and numerical simulation can be carried out.

The remainder of this paper is outlined as follows: the process of numerical simulation is analysed and the overall strategy used for the work is presented. Knowledge structures are detailed in the context of frame-based knowledge representation. The inference process and integration of the system with the existing PDE solver PDEQSOL is discussed.

2 An Overview of the AI-PDEQSOL System

2.1 Problem Analysis

In order to establish a proper approach to deal with the problems mentioned above, a careful analysis of the process of numerical simulation has to be made. Figure 1 demonstrates such a process between a real world problem and a numerical model by illustrating a user's problem of the cooling of an electronic device. The definition of these levels are given here.

Real world problem

A real-world problem normally consists of a number of physical phenomena. In the problem, the phenomena may be coupled with each other and the region may take three dimensional complicated shapes. An example is given in Figure 1a. This level is not dealt with in this paper.

Physical Model

The physical model is a simplified and precise representation of a real-world problem, which is considered to be the principal input level of AI-PDEQSOL. The contents are as follows:

1. The names of physical phenomena and boundary types
2. The regions over which each physical phenomenon takes place
3. The geometric descriptions of these regions
4. Source effects
5. Initial state of each physical phenomenon which requires transient analysis.

Figure 1b illustrates a possible physical model for a cooling electronic device.

Mathematical Model

The mathematical model consists of a set of PDEs and boundary conditions which represents mathematically the behaviour of the physical phenomena. The contents are as follows:

1. PDEs with source terms, boundary conditions and material properties.
2. Initial conditions for each time-dependent PDE.
3. The region definitions corresponding to each PDE.

Figure 1c illustrates the mathematical model of a cooling electronic device.

Numerical Model

The numerical model contains all the information necessary to drive a PDE solver. Also a spatial mesh and a time step must accompany each schematic description. Figure 1d shows an example of the numerical model using the syntax of PDEQSOL.

2.2 System structure

AI-PDEQSOL has been realised based on a model transformation approach. A physical model is taken as an input and converted to a mathematical model, then to a corresponding numerical model, which is outputted in the form of a PDEQSOL program. These two stages of transformation are carried out using two knowledge-bases; a Mathematical knowledge-base and a Numerical knowledge-base. The overall structure of AI-PDEQSOL is shown in Figure 2. Each transformation process is outlined as follows.

Transformation of physical model to mathematical model

1. The physical model is defined through the user interface data.
2. A set of PDEs and boundary conditions is created from the Mathematical knowledge-base.
3. Values are assigned to all the material properties referenced in each PDE and boundary condition. They are retrieved from the material properties database.

Transformation of mathematical model to numerical model

1. A linearisation algorithm, a time expansion technique and a matrix inversion technique are chosen from the Numerical knowledge-base.
2. A single time-step and a spatial mesh division are calculated to satisfy numerical constraints
3. A PDEQSOL source code corresponding to a numerical model is generated.

Since a number of numerical models can be equally valid (one may be more accurate but less efficient than another), AI-PDEQSOL generates several possible numerical models. The user is consulted to take the final decision about which model is preferable for the actual simulation.

3 Knowledge Representation

This section deals with how knowledge is represented in the system. Knowledge can be classified as follows.

1. Domain knowledge includes mathematical/numerical knowledge for describing problems.
2. Characterisation knowledge allows a given problem description to be characterised so that the components of a technique for its solution can be extracted from the domain knowledge.
3. Constraint knowledge is represented by constraint rules which must be satisfied if a particular solution technique is to give a satisfactory result.
4. Evaluation knowledge is used to estimate and rate the features (efficiency and accuracy) of the numerical models derived from the transformation process.

The use of frame-based knowledge representation built using the object oriented paradigm has been fully exploited to give a concrete form to the above knowledge. They are discussed in the following sections in detail.

3.1 Domain Knowledge

Domain knowledge is stored mainly in the two knowledge-bases.

Mathematical Knowledge-base

The Mathematical contains all necessary knowledge to form PDEs and boundary conditions. The knowledge-base consists of a class hierarchy, which defines the templates of objects of which any mathematical model is composed. The five primary classes are Phenomenon, PDE Group, PDE, Term and Variable. As shown in Figure 3, these classes give a hierarchical representation of a PDE definition. Each main class has the subclass structure shown in Figure 4. For example, the Term subclasses are defined on the basis of terms which share common differential operators such as div[(.)grad(.)].

Knowledge associated with each class is stored as either structural links or localised rules bound to the class. Using the rules, the appropriate instantiations from the knowledge-base classes (i.e. components of a PDE group) are inferred during the first transformation. The structural links, which correspond to the object (part-of) hierarchy between the instantiated objects, are also stored in each class definition to relate the instantiated objects and to form a PDE group. This mechanism is discussed in Section 4.1.

The Mathematical Knowledge-base also contains the knowledge required to set source terms and boundary conditions for a PDE. Source terms are considered in the same way as other terms except that the rules associated with them are related to the occurrence of two or more phenomena in the given physical model. Boundary conditions are associated with each PDE class definition by structural links.

Numerical knowledge-base

There are three substructures in the Numerical knowledge-base: Non-linear algorithms for linearising PDEs, time Expansion algorithms for dealing with time-dependent PDEs, and Matrix Inversion algorithms for solving the matrix resulting from discretisation of the PDEs.

The algorithms are closely related to the PDEs and therefore the objects in the knowledge-base are linked from PDEs by a 'parts-of' link. The structure is illustrated in Figure 5. As with the Mathematical knowledge-base, each algorithm class definition in the knowledge-base stores localised rules. The rules are concerned with algorithm selection and contain the knowledge necessary to link the PDEs in a mathematical model to the appropriate algorithm. By these rules, connections between the characteristics of the PDE problem under consideration and applicability of each algorithm are measured. For example, the following rule is stored associated with ILUCG algorithm:

IF the *PDE is symmetric* THEN *select ILUCG Method*

3.2 Characterisation Knowledge

The antecedents of localised rules mentioned above relate to characteristics of the problem, rather than to specific parameters. For example, in the first transformation, fluid flow must be characterised as turbulent or laminar, or, in the second transformation, the PDE must be characterised as symmetric or asymmetric. This is achieved using characterisation knowledge. This knowledge resides in the physical model and mathematical model class definitions and is currently implemented as class methods bound to each model. The required information is determined by calculating a characteristic number and comparing it with a threshold value.

3.3 Constraint Knowledge

Numerical solutions of PDE problems must obey certain well-defined numerical constraints. Bounds on the spatial mesh size and the time step must be satisfied in order to obtain a stable and accurate solution. In addition to numerical constraints, the user may supply personal constraints such as a maximum CPU time or computer memory available.

This type of knowledge is represented as methods bound to the numerical model class definition. Given an algorithm set derived from Numerical knowledge-base, these methods return the appropriate spatial mesh and time step under the numerical and personal constraints. When suitable mesh size or time step can not be determined within the constraints, the algorithm set under consideration is abandoned and the next possibility is examined. This mechanism is discussed in Section 4.2.

3.4 Evaluation Knowledge

The system generates a number of possible numerical models at the end of the transformation process. Since each numerical model contains complete information to carry out numerical simulation, it is possible to estimate its features, such as memory requirements, accuracy and CPU time. In a similar way to the constraint knowledge, evaluation knowledge is stored as methods bound to the numerical model class definitions.

4 Inference Process

The inference mechanism, the creation of a solution space and generation of PDEQSOL code are discussed in this section.

4.1 Selection Mechanism

The essential part of the inference system in AI-PDEQSOL is a global selection mechanism which acts over the hierarchical knowledge representation in the two knowledge-bases. The selection mechanism scans the object hierarchy from the top to the bottom by referring to the potential structural link specified in each class definition. A simple example of the action of the selection mechanism is shown in Figure 6. Assume that the selection mechanism is working at Object A which has been instantiated from Class A in a knowledge-base. The potential structural links from Class A to lower level component classes are known and stored as domain knowledge. The selection mechanism refers to these links and activates the localised rules associated with Class B, C, D and E in turn. Each rule determines whether the corresponding class can be instantiated under the given condition. If Class C and E are determined as usable, the instantiations (i.e. Object C and E) are created and the actual 'part -of' links are established with Object A.

For example, the class definition of the diffusion-advection-equation contains the potential structural links to its components: time-term, advection-term, diffusion-term and source terms. If the physical model under consideration is characterised as transient, pure-diffusion and non-coupling, only the time and diffusion terms are instantiated and linked with the diffusion-advection-equation object.

4.2 Creation of a Solution Space

The process of mathematical model to numerical model transformation involves several decision stages as described in Section 3.2. These stages are closely related to each other and cannot be made in a simple serial fashion. Referring to Figure 7, in the first three stages, by applying domain knowledge in the algorithm knowledge-base to the chosen PDE, the selection mechanism works so that more than two possibilities can be generated at each decision stage. In the time step and mesh division stages, constraint knowledge is applied and a mesh pattern and a time step are set for each branch. Some of the solution branches may not satisfy certain constraints and are therefore abandoned as dead ends. When a complete solution space is created, the methods which encapsulate the evaluation knowledge are invoked and the features of each solution are estimated.

4.3 Creation of Numerical Simulation Code

When one numerical model has been chosen by the user, the system invokes the code generation methods which are bound to the algorithm objects of the selected numerical model. Each algorithm object holds a code generation method particular to the algorithm. A series of these code generation methods expands the PDEs in the numerical model into a PDEQSOL source code, which can then be executed by the PDEQSOL system.

5 Implementation and Current Status

A prototype of the system has been implemented on a Macintosh II using Object Lisp, an object-oriented extension of Common Lisp. A graphical interface is provided in order to help the user to define the physical model. The simulation results can also be monitored through the interface.

The system currently deals with problems in the heat transfer and fluid domains. The applicable geometry is limited to two dimensional problems. As the discretisation method, only Finite Element Method can be used. The knowledge-bases are currently developed for experimental purposes and contain several typical algorithms and constants for each domain.

6 Conclusion

An AI based problem solving environment for numerical simulation of the problems described by partial differential equations (PDEs) has been presented. Transformation between three abstract models (physical, mathematical and numerical models) is realised using tailored knowledge-bases. In order to build these knowledge-bases, the applicability of frame-based knowledge representation has been investigated so that they reflect the hierarchical nature of the domain knowledge. Also the combination of localised rules and a global inference mechanism has been realised on the basis of the object oriented paradigm.

Future developments will involve extensive expansion of the knowledge-bases and rules so that the system will be able to deal with more complex coupled problems over irregular shaped regions.

Acknowledgements

The authors would like to express their appreciation to Dr. J.B. Grimson and Professor J.G. Byrne of Trinity College, Dublin for their advice and encouragement. We would also like to thank N. Hataoka of Hitachi Dublin Laboratory for his helpful comments.

References

[Barras et al 1990] Barras, P., Blum, J., Paumier, J.C., Witomski, P. and Rechenmann, F. "EVE: An Object-Centred Knowledge-Based PDE Solver". In *Proceeding of the Second International Conference on Expert Systems for Numerical Computing,* Purdue University, USA, 1990.

[Chelsom et al 1990] Chelsom, J., Cornali, D. and Reid, I. "Numerical and Statistical Knowledge-Based Front-ends, Research and Development at NAG". In *Proceeding of the Second International Conference on Expert Systems for Numerical Computing,* Purdue University, USA, 1990.

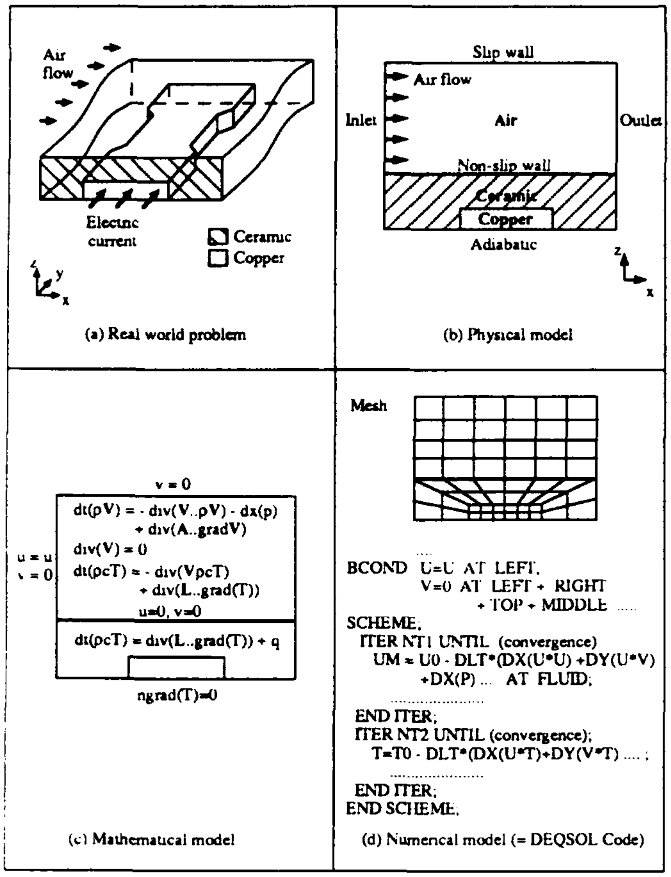
[Kon'no et al 1986] Kon'no, C., Saji, M., Sagawa , N. and Umetani, Y. "Advanced Implicit Solution Function of DEQSOL and its Evaluation". In *Proceedings of IEEE Fall Joint Computer Conference,* pp.1026-1033, Dallas, USA, 1986.

[Rice 1985] Rice, J.R. "ELLPACK - An Evolving Problem Solving Environment for Software Computing". In *Proc.of IFIPTC2/ WG 2.5 Working Conference on Problem Solving Environments for Scientific Computing,* pp. 233-245, 1985.

[Russo et al 1987] Russo,M., Peskin, R., Kowalski, A. "A prolog-based expert system for modeling with partial differential equations". In *Simulation, Vol 49, No.4,* pp.150-158, 1987.

[Schonauer and Schnepf 1987] Schonauer, W. and Schnepf, E. "Software Considerations for 'Black Box' Solver FIDISOL for Partial Differential Equations". In *ACM Trans. on Maths. Soft., Vol 13, No. 4,* pp. 233-245, 1987.

[Umetani et al 1985] Umetani, Y. "DEQSOL - A numerical Simulation for Vector/Parallel Processors". In *Proc. of IFIPTC2/ WG 2.5 Working Conference on Problem Solving Environments for Scientific Computing,* pp. 147-164, 1985.



(a) Real world problem

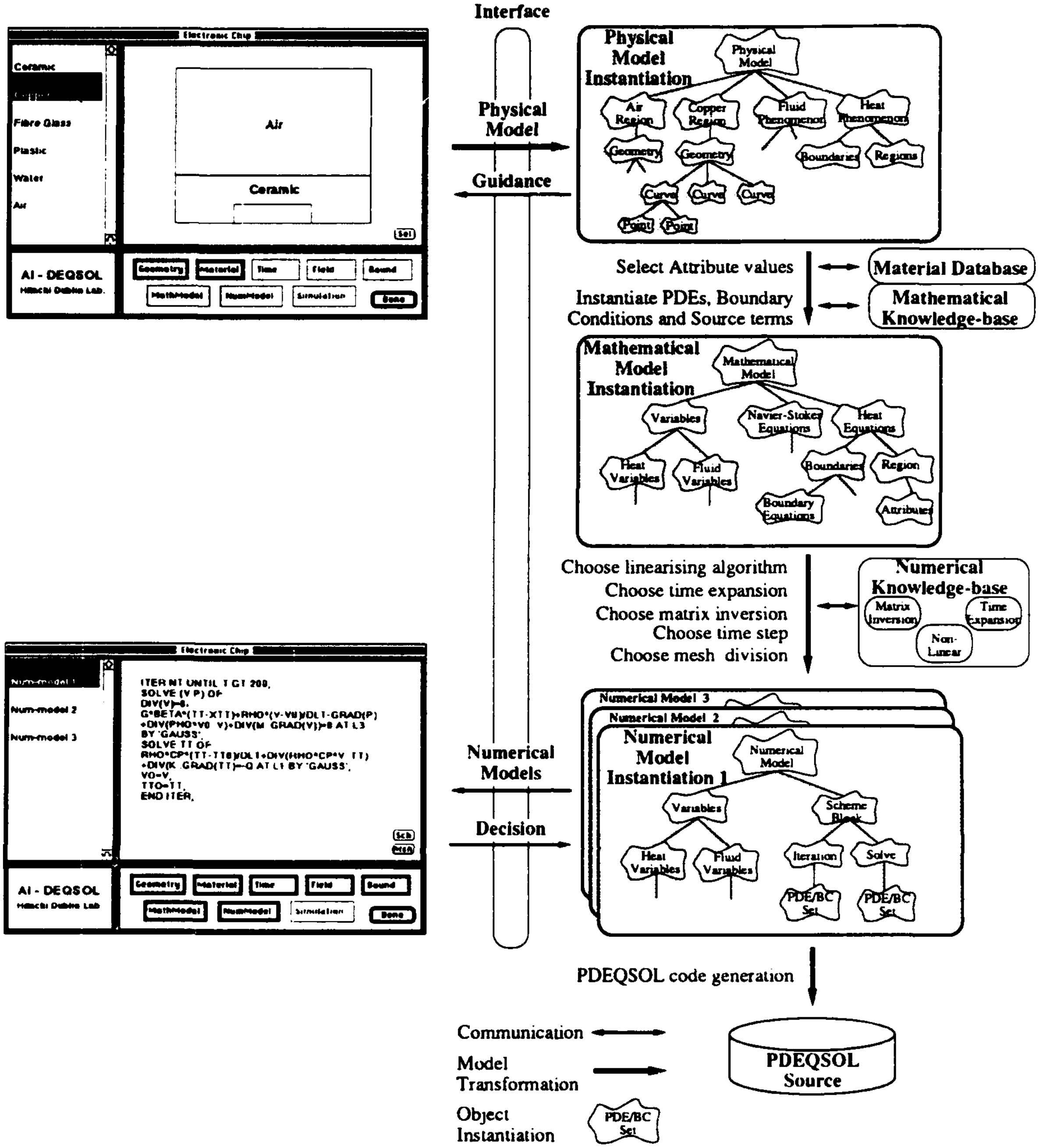
(b) Physical model

(c) Mathematical model

```
          ....
BCOND  U=U  AT  LEFT,
       V=0  AT  LEFT +  RIGHT
                + TOP + MIDDLE  ....
SCHEME;
  ITER NT1 UNTIL  (convergence)
    UM = U0 - DLT*(DX(U*U) +DY(U*V)
          +DX(P) ...  AT  FLUID;
      ......................
  END ITER;
  ITER NT2 UNTIL (convergence);
    T=T0 - DLT*(DX(U*T)+DY(V*T) .... ;
      ......................
  END ITER;
END SCHEME;
```

(d) Numerical model (= DEQSOL Code)

Figure 1 Modelling a cooling LSI chip



```
ITER NT UNTIL T GT 200,
SOLVE (V P) OF
DIV(V)=0,
G*BETA*(TT-XTT)+RHO*(V-V0)/DLT-GRAD(P)
+DIV(RHO*V0 V)+DIV(M GRAD(V))=0 AT L3
BY 'GAUSS',
SOLVE TT OF
RHO*CP*(TT-TT0)/DLT+DIV(RHO*CP*V TT)
+DIV(K GRAD(TT))=Q AT L1 BY 'GAUSS',
V0=V,
TT0=TT,
END ITER,
```

Figure 2 AI-PDEQSOL System Structure

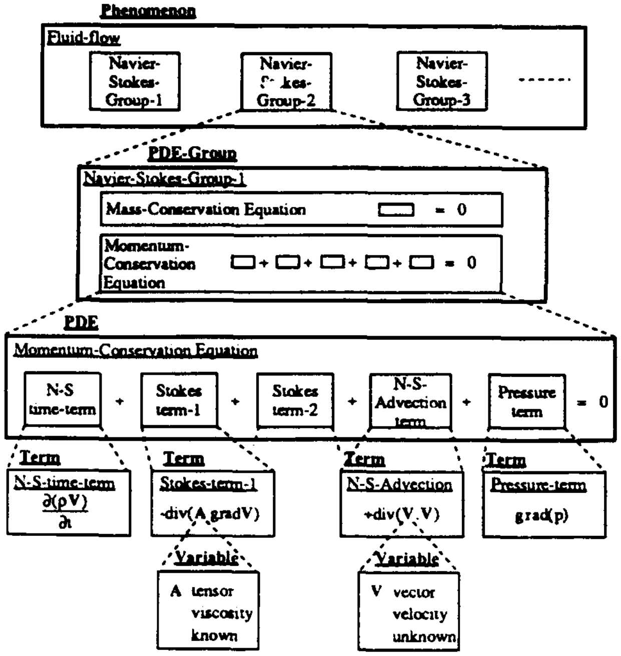


Figure 3 Hierarchical PDE Representation

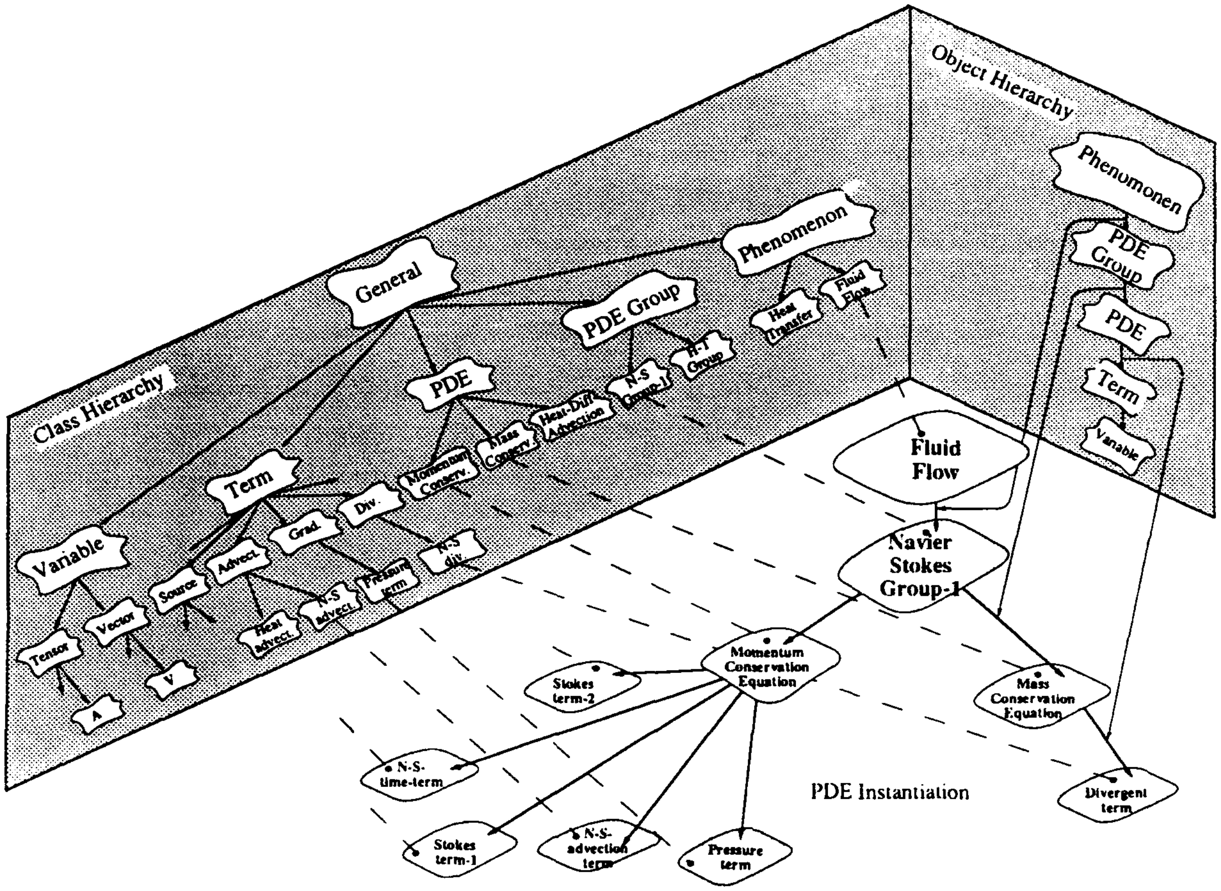


Figure 4 PDE/BC Database Representation

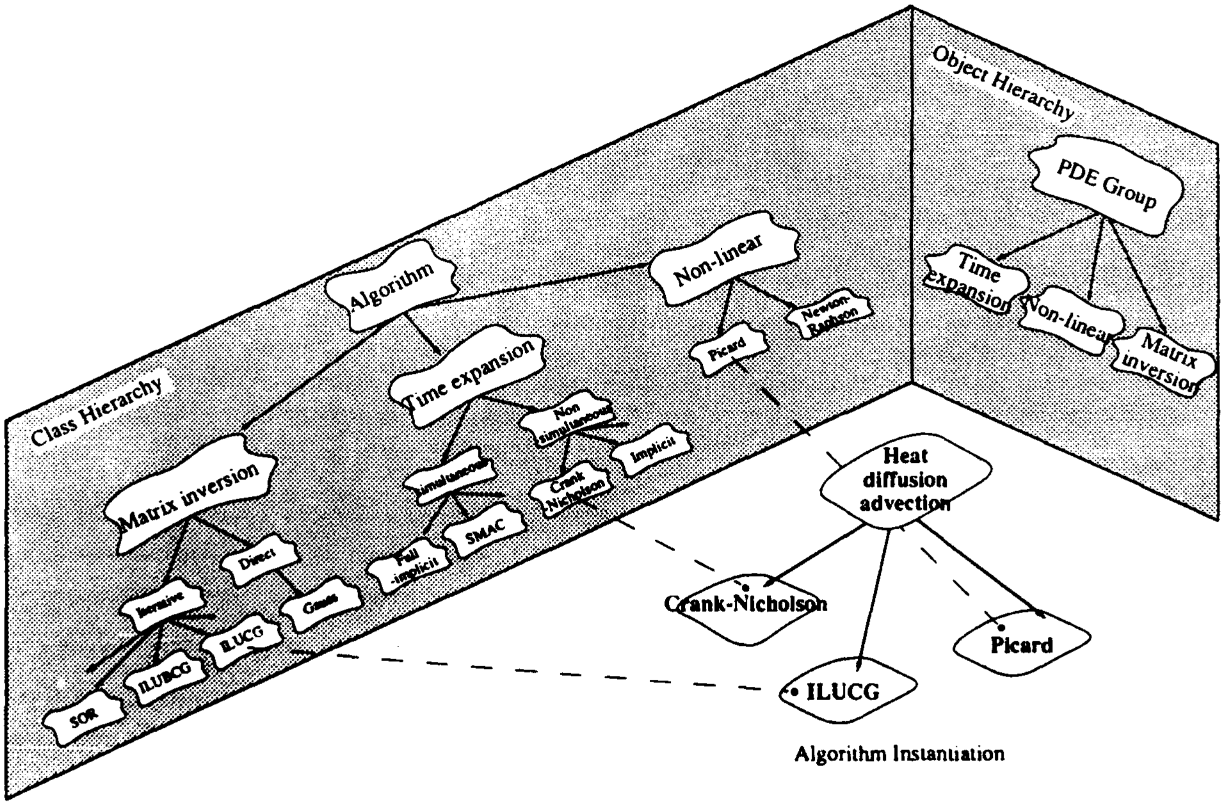


Figure 5 Algorithm Database Representation

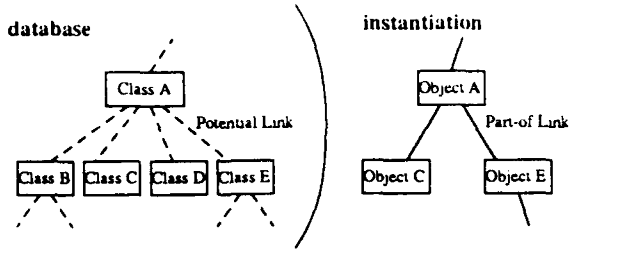


Figure 6 Selection Mechanism

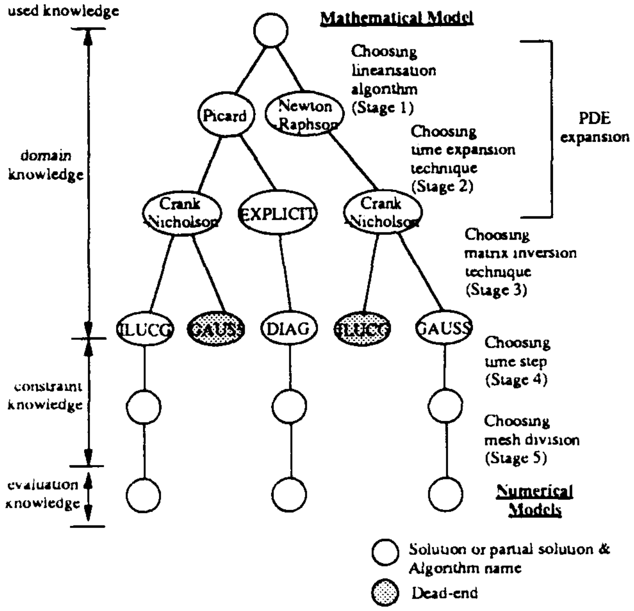

used knowledge
domain knowledge
constraint knowledge
evaluation knowledge
Mathematical Model
Choosing linearisation algorithm (Stage 1)
Picard
Newton Raphson
Choosing time expansion technique (Stage 2)
PDE expansion
Crank Nicholson
EXPLICIT
Crank Nicholson
Choosing matrix inversion technique (Stage 3)
ILUCG
GAUSS
DIAG
ILUCG
GAUSS
Choosing time step (Stage 4)
Choosing mesh division (Stage 5)
Numerical Models
Solution or partial solution & Algorithm name
Dead-end

Figure 7 Creation of Solution Space

Expert Systems as an Intelligent User Interface for Symbolic Algebra

by

Mike Clarkson
Centre for Earth and Space Science
York University, North York
Ontuio, M3J 1P3, Canada

Abstract

General purpose computer algebra (CA) system such as Reduce, Macsyma, SMP, Maple and Mathematica, have been in widespread use in academia for more than a decade, and have found a wide range of applications. Many of these systems have a sophisticated graphics interface, and contain encyclopedic databases of knowledge about mathematical functions and techniques. Some systems allow the user to solve part of the problem symbolically, and then generate FORTRAN code to evaluate the results efficiently. Other systems can be linked to object libraries of numerical code such as the NAG library, to extend their range to include numerical techniques as well as symbolical techniques.

But despite their widespread use and wide range of applicability, our observation of the users of all of these systems is that they have proven to be very difficult to use. Rarely have we received the complaint that the systems were not powerful enough, or incomplete. In many cases, the users of these systems were frustrated because they were unaware of the proper command that they should use, or they were trying to solve their problem with brute force, without applying any of the finesse that mathematics requires. But without a doubt, the biggest hurdle has been the difficulty of learning how to use these systems.

Perhaps one of the worst systems from a user's point of view is Macsyma, the symbolic and algebraic manipulation system developed at MIT during the 1970's. Macsyma is a very large CA system written in Lisp, with over 1000 global variables and functions defined and documented. Macsyma was developed by many people over a ten year period, and grew without the benefit of a coherent naming scheme for the functions and variables. But beyond these specific problems of the program itself, some of the difficulty associated with using these systems arises from the lack of semantic uniqueness in mathematics. For example, factoring an expression usually reduces it in size:

$$\mathrm{FACTOR}(x^6 + 3x^4 + 3x^2 + 1) \rightarrow \qquad (x^2 + 1)^3$$

However, in some cases factoring an expression makes it bigger:

$$\mathrm{FACTOR}(x^6 - 1) \rightarrow \qquad (x - 1)(x + 1)(x^2 - x + 1)(x^2 + x + 1)$$

Thus the intuitive concept of "reducing" or making smaller does not always correspond to a unique mathematical counterpart. Furthermore, the most straight forward approach from a mathematical point of view may be less than optimal from a CA point of view. Often the former might be characterised as a "brute-force" approach, lacking in any kind of subtlety or optimization. Although provably correct, the mathematical approach may not be successful on a CA system because of limitations of time or memory resources. This is especially critical in computer algebra, because many of the exact algorithms grow exponentially or even factorially with increasing size. This internal "expression swell" can be fatal to large brute-force computations.

In fact, the skills needed to operate an "expert system" like Macsyma are in themselves a considerable body of knowledge. One has to have good understanding of mathematics to know how to approach any non-trivial problem, as well as extensive training on how to implement the chosen approach. Further consideration must also be given to selecting implementation strategies based on efficiency. Given the structured nature of this knowledge and expertise, it suggests the idea of combining CA systems with rule-based expert systems in order to provide an intelligent guide to aid in mathematical problem solving.

We have implemented an intelligent user interface for the symbolic algebra system Macsyma. The user interface is intended for beginning to intermediate users who at present are often daunted by the wide range of commands and capabilities of a symbolic algebra system. The expert shell acts as a menu-driven user interface to the computer algebra system, and organizes the vast mathematical range of a computer algebra system into a useable structure. It performs source level optimizations in order to plan the most efficient solution strategy for the computer algebra system. It has been integrated with a relational on-line help system, that is organized similar to a college algebra text.

The system begins by asking the user a series of questions to try to define what operation in mathematics he or she wishes to carry out. As the domain of the operation becomes increasingly specified, the knowledge tree is descended instantiating the required sub-frames, and the questions become increasingly more specific. The system analyses the expressions under consideration, and generates the Macsyma code needed to solve the problem. It then communicates the code to a back-end Macsyma sub-process.

The system has been written using an Emycin rule-based expert system, and has been implemented in Common Lisp and CLOS under X-Windows.

DESIGN AND IMPLEMENTATION OF SYMBOLIC COMPUTATION SYSTEMS

Alfonso Miola

Dipartimento di Informatica e Sistemistica
Università degli Studi di Roma "La Sapienza"
Via Salaria 113, 00198 Roma, Italy.

Abstract

The paper presents the development of a research project based on a new methodological approach to the design and implementation of a symbolic computation system for mathematical problem solving. The proposed symbolic computation system allows one to compute by formal algebraic and analytical methods and to prove properties of the computation by automated deduction mechanisms. The design and implementation is based on the object oriented programming paradigm.

1. Introduction

The research and development in Symbolic Computation has brought to the design and implementation of software systems, such as DERIVE, MACSYMA, MAPLE, MATHEMATICA and REDUCE, widely available since several years. These systems have been successfully used in several application areas of sciences and engineering [Buc83, Cav86, Dav88, Mio90a, Mio90b, Mio91,Yun80].

Unfoıtunately, those systems cannot always support the qualitative activity of mathematical problem solving, where often the needs are for analysing the mathematical properties of computed results.

Actually, even very simple computations, carried out by those systems, can produce results which are not acceptable from a mathematical point of view. Let us present some

Research partially supported by MURST under the projects: "Calcolo Algebrico", "Metodi e strumenti per l'elaborazione non numerica"; and by CNR under the project: "Sistemi Informatici e Calcolo Parallelo".

examples of these incorrect computations, with no reference to any specific symbolic computation system. As a matter of fact, the problems we want to discuss are very general in symbolic computation and they are mainly due to the design features of the software systems, generally based on unsafe programming methodologies.

Example 1

$$(\sqrt{x})^2 \longrightarrow x; \qquad \sqrt{x^2} \longrightarrow x.$$

Example 2

$$Let\ y\ be\ \frac{1}{x}; \qquad Let\ z\ be\ \frac{d}{dx}\, log\, x; \qquad y - z \longrightarrow 0.$$

Example 3

$$Let\ y\ be\ x^i; \qquad \sum_{0 \leq i \leq 10} y \longrightarrow 11x^i.$$

Example 4

$$Let\ y\ be\ \sum_{i=1}^{n} x^i; \qquad \frac{\partial y}{\partial x_1} \longrightarrow 0.$$

Example 5

$$\infty - \infty \longrightarrow 0; \qquad \infty + \infty \longrightarrow 2 \cdot \infty; \qquad \infty \cdot 0 \longrightarrow 0.$$

The examples 1 and 2 are incorrect from a mathematical point of view. The function $(\sqrt{x})^2$ and the function $\sqrt{x^2}$ are defined in two different domains. The same for the functions $\frac{1}{x}$ and $\frac{d}{dx}\, log\, x$, even if they have the same analytical expression $\frac{1}{x}$. A system simply designed on the basis of syntactic rewriting rules usually cannot take into consideration any of the qualitative mathematical attributes of a syntactic object. The examples 3 and 4 are wrong for completely different reasons: the symbol i is used with different meanings in the two successive steps of each example. The example 5 shows the obvious need for a precise algebra and therefore for an adequate semantics to deal with infinity quantities.

More in general, the use of a system designed without appropriate semantics is a main reason for the errors detected in all the examples presented above.

Example 6

$$\sum_{-1\leq i\leq 10}\int x^i dx \longrightarrow \sum_{-1\leq i\leq 10}\frac{x^{i+1}}{i+1} \longrightarrow \ ?$$

In general, a message like "division by zero" occours with this computation. This example again shows an inappropriate use of the symbol i, in the integrand x^i, which is not correctly bound to the constraint imposed in the sum expression. In order to avoid this situation, the computation should proceed by interchanging the sum operator with the integral sign and then dividing the sum into parts, as follows.

$$\sum_{-1\leq i\leq 10}\int x^i dx \longrightarrow \int \sum_{-1\leq i\leq 10} x^i dx \longrightarrow$$

$$\longrightarrow \int (x^{-1} + \sum_{0\leq i\leq 10} x^i)dx \longrightarrow log\, x + x + \frac{x^2}{2} + \cdots + \frac{x^{11}}{11.}$$

None of the existing symbolic computation systems is able to complete this computation, and particular skill is required to the user to solve this problem.

2. Specification and manipulation of mathematical objects and methods.

On the basis of these considerations a research project has been proposed to identify a new methodological approach to the design and implementation of symbolic computation systems. An symbolic computation system TASSO is under development to experiment with the proposed methodology. The project deals with abstract entities as objects described by axiomatic specifications. It considers logic formulas and algebraic structures. Each object has a unique formal definition with the specification of its attributes and algebraic, analytical, logic computation are possible under fixed constraints.

In particular, the project is mainly based on the following aspects:

- programming methodologies for abstract specifications;
- axiomatic definitions of mathematical objects;
- automated deduction mechanisms, as a new basic computing tool;
- algebraic and heuristic methods for applied mathematics, at a high level of abstraction with respect to the domains where the problems are defined.

The specification of an object is given by following the typical object oriented mechanism of inheritance, so as to reproduce the classical hierarchy of the mathematical abstract structures. For instance, the abstract algebraic structure *Ring* is specified by assuming the specification given for the structure *Group*, as in abstract algebra.

The dynamic definition and manipulation of mathematical objects is also possible: the specification of an object (e.g. matrix, polynomial) is unique and the instantiation mechanism allows one to compute with objects defined at run time (e.g. matrix over polynomials over ... matrices of integers).

Moreover, the specification of an object includes the specification of admissible computing methods, given through an abstract definition at the highest possible level of the object hierarchy. For instance, the *Euclidean Algorithm* it is specified as an attribute of the object *Euclidean Domain*. Then the instantiation of the *Euclidean Domain* implies the instantiation of all the related computing methods, (e.g. the *Euclidean Algorithm*).

From an application point of view both numerical and non-numerical computing methods are needed in a software system for symbolic computation. Actually, the introduction of numerical methods can be done consistently with the methodological approach of the entire project, so as to offer the two types of computing methods in a single integrated computing environment.

In order to develop such an integrated environment, one starts from the well known similarity of numbers and polynomials, with their related arithmetic operations, and builds very general abstract data structures to encapsulate various similar objects. At the same time, the algebraic p-adic construction (Hensel method) and the classical numerical approximation method (Newton method) can be viewed as special cases of a more general algebraic approximation method in abstract structures.

Then, the available specification mechanism allows one to exploit the main results on the integration and the *amalgamation* of numerical and algebraic methods [Lim90, Lim91, Mio88a, Mio90a, Mio90b]. The concept of amalgamation is borrowed from logic programming and expresses the methodological approach to software development based on two levels of operations specified with uniform semantics. The operations of the upper level control the execution of the operations at the lower one. Actually, numerical computing can be interpreted as a lower level where the computations are completed under the control of symbolic computing at the upper level, where the complete formalization of the entire computation is given.

Furthermore the recent result on the possible unified interpretation of the Hensel construction for algebraic equation solving and of the Buchberger method for the solution of systems of polynomial equations, could support this definition of a general method for abstract approximation construction [Mio88b].

The inheritance mechanism in specifications is strictly related to the notion of subtyping, as any derivation obtained by inheritance corresponds to a compatible assignement rule. Furthermore, the subtyping relation is strictly connected to the functionalities available through the definition in a class as export parameters. Then, subtyping is a more general relation than inheritance, and it is possible to specify when a class can be assumed as subtype of another class [Par91].

The problem of automatic execution of specifications is also considered in the project. A particular model is proposed for the specifications, as sorts, signatures and axioms which give meaning to the function symbols. The verification of properties of given objects is obtained by two different mechanisms: verification "a priori" and verification "a posteriori". In the case of "a priori" verification an editor oriented to the construction of the specification has been proposed. Executable code is then derived from the edited specifications [Ant90a, Ant90b]. For the "a posteriori" verification, a completeness checker for the rewriting rules associated with the specifications is available [Foc90].

The system TASSO includes two main modules: TASSO-L and TASSO-D. The TASSO-L is the language, based on an object oriented approach, which allows the user to define objects, to instantiate abstract structures into actual computing domains, and to compute with available objects.

The TASSO-D is the module for automated deduction. It can be used both by the user to check properties of given objects and by the system itself in order to generate and derive new properties of given objects from properties already known.

3. The language TASSO-L.

On the basis of the characterization given for the specification process, the object oriented programming paradigm has been recognized as the most natural. Actually, the object oriented approach can be well characterized by the following equation

$$\textit{Object-Oriented} = \textit{ADT} + \textit{Inheritance}$$

according to the literature (e.g. [Str87]). In fact, all the mathematical objects under consideration, can be easily modelled by ADT. Furthermore, their implementation is encapsulated at a hidden level respect to the user. Then, mathematical objects represented by ADT are viewed as generic structures in which data, methods and attributes are specified and localized.

The available inheritance mechanism gives the possibility to specialize or extend ADT, and the typical hierarchy of mathematical objects is made correspondent to the tree of the inherited structures, and it becames embedded into the system. Moreover, a correct cooperation between abstraction and inheritance allows also to obtain *parametric polymorphism.*

Following this approach, the common properties of similar data structures are defined at the highest possible level of abstraction and the methods to perform specific operations are dynamically activated only upon the appropriate operands.

According to the original motivation of the research project, one of the most important characteristics to be considered is the necessity to mantain a high degree of correctness of the performed operations, also when dynamic data are defined: i.e. the flexibility of the language respect to user's needs must be balanced by preventing incorrect or ambiguous types definitions. This fundamental objective can be obtained by a correct regulation of the instantiation mechanism of the dynamic data. At the same time, the mechanism for strong type checking, acting on all the defined objects, is the key to support the correctness of the types and subtypes defined and used in all the different steps of the computation.

We have considered the most interesting, correct and flexible features of the existing object oriented programming languages [Lim90]. We have also been experimenting with some of them, namely Eiffel [Mey88], Smalltalk [Ing78], C++ [Str86] and Loglan [Kre90].

On the basis of these experiments we have chosen Loglan for a prototype implementation, also because it refers to the theory of Algorithmic Logic [Cio89, Mir87] as the main theoretical framework where the specification and the correctness verification can be carried out.

The language allows the objects to be connected in a tree structure by a multi-level *prefixing*, and the attributes of the objects are dynamically defined and redefined by *virtual* specifications.

The inheritance mechanisms which have been selected are those of strict and multiple inheritance. The strict inheritance allows relations of *generalization*, while the multiple inheritance allows one to derive properties of an object from those of more than one father, obtaining a kind of *aggregation.* Those two mechanisms offer a good level of correctness and flexibility, as described in [Reg90a, Reg90b, Reg90c].

4. Automated deduction mechanisms.

The language TASSO-L incorporates automated deduction mechanisms of the module TASSO-D. Those mechanisms can be activated directly by the user when he has to accomplish some steps of deduction, and they are also automatically applied by the system itself as a tool to guarantee the correctness of the objects definitions.

The mathematical object to be manipulated are specified by axioms in the first order logic formalism. The problems of logic property manipulation can be classified according to different operational modes and to computing needs of users.

In particular, in some cases it is necessary to verify the validity of specified attributes of an object or of a class of objects. This case is named *verification mode*. In same other cases, properties can be generated from known properties already verified. This is the case of *generation mode*. Furthermore, sometimes, it is necessary to abduct the premises from which given valid properties can be generated. This is the case of *abduction mode*.

These various needs can be satisfied by a unique deduction method.

The TASSO-D module incorporates different deduction mechanisms. In the present version of TASSO, object-oriented implementations of the *Resolution* with different kinds of *Backtracking* [Bon88], of the *Connection Method* [Bib87], [For90] and of a *Sequent Calculus* [Gal86], [Bon90] are available.

These mechanisms offer a good degree of human orientation and in some cases their applicability is increased by the interactive use. The user has the knowledge of his application field, can modify the deduction path or stop it or, also, execute it under the control of a specific ad-hoc strategy. Therefore the interaction is useful in this perspective.

The Connection Method is based on the possibility of representing *w.f.fs.* as matrices whose elements are (matrices of) atoms, and a validation of a formula is obtained by searching a path of connections between columns of the matrix, which represent the clauses of a NDF of the formula. Then, this method can be defined as an algorithmic mechanism operating upon a single data structure which directly corresponds to the given formula.

Some remarks can be made on the characteristics of this deduction mechanism.

- The efficiency and the transparency of the method derive from the possibility of working, during the entire deduction process, with the same unmodified matrix, created in the input step. In particular it has to be stressed that any kind of preprocessing of the original formula (e.g. to transform it in a normal form) can be avoided.
- This mechanism can be considered flexible because it is extendible to higher order logics and also because the corresponding proof based on natural Gentzen calculus can be automatically obtained from a proof based on this Connection Method.
- The Connection Method, using a single data structure, in which the formula is completely encapsulated, is particularly suitable of an object-oriented implementation and in this way the uniformity with the entire system can be maintained.

The extension of the deduction capabilities to include the *Abduction Rule* in the *Resolution* approach (as proposed by [Kow83]) is known in the literature. Similarly the extension from the verificative case to the causes for events case has been proposed for the Connection Method [For89].

Furthermore the possibility of defining a single method to support different kinds of deductions, namely verificative, generative and abductive, has been considered. To this purpose, a *Sequent Calculus* has been defined. This method results to be intrinsecally suitable by being based on a set of rules easily defined and applied according to the different objectives.

The sequent mechaniśm has well known advatages: it is a natural deduction mechanism, it can be used interactively and monitored by the user, and it is particularly flexible by allowing one to transform the deduction rules according to the nature of the problems to be solved.

In [Bon89] e [Bon90] a sequent calculus has been proposed including the three deduction modes, namely, verification, generation and abduction. Particular attention has been devoted to the termination problem.

Starting from the Gallier's proposal [GA86], the sequent calculus has been defined with a set of rules for both the decomposition of a given formula into elementary atoms, and for the composition of many sequents into a normal form. The propositional case represents the kernel of the tool to be also used in the predicative case, following the object oriented approach for its implementation. We have used some strategies in order to augment the efficiency and to give some halting criteria to overcome the undecidibility of the general problem.

5. Development and testing.

The starting point of this experimentation has been proposed by D. Wang stating a set of interesting problems together with some solution strategies [Wan90]. Those examples came from different areas of applied mathematics, such as irrational expressions simplification, stability of differential equations, linear algebra, limits, geometry reasonings and number theory.

According to the original motivation for the project all these examples might be impossible to be treated by the existing systems for the mathematical problem solving and can be assumed as a good bed for the test of the system.

The development of the project has followed an incremental approach. Once a preliminary version of the language and the deduction mechanisms has been made running we started by defining some classes of elementary objects, such as integers, rationals, polynomials, matrices.

During this developing phase the automated deduction mechanism available have been widely used to check properties of objects, to generate properties of objects under definition, to verify the correctness of the computing mechanism encapsulated into the objects.

References.

[Ant90a] Antoy, S., Design Stategies for Rewrite Rules, Proc. 2nd Int. Workshop on Conditional and Typed Rewriting Systems, (1990).

[Ant90b] Antoy, S., Forcheri, P., Molfino, M.T., Specification-based Code generation, Proc. Hawaii Int. Conference on System Sciences, 165-170, (1990).

[Bib87] Bibel, W., Automated theorem proving, II Ed., Fried. Vieweg & Sohn, Braunschweig/Wiesbaden, (1987).

[Bon89] Bonamico, S., Cioni, G., Embedding flexible control strategies into object oriented languages, LNCS 357, Springer Verlag, (1989).

[Bon90] Bonamico, S., Cioni, G., Colagrossi, A., A Gentzen based deduction method for the propositional theory, Rapp. IASI 289, (1990).

[Buc83] Buchberger, B., Collins, G.E., Loos, R.G.K., (eds.), Computer Algebra - Symbolic and Algebraic Computation, 2nd ed., Springer-Verlag, (1983).

[Cav86] Caviness, B.F., Computer Algebra: Past and Future, J. Symb. Comp., 2:217-236, (1986).

[CS89] Cioni, G., Salwicki, Advanced Programming Methodologies, Academic Press (1989).

[Dav88] Davenport, J.H., Siret, Y., Tournier, E., Computer Algebra - Systems and Algorithms for Algebraic Computation, Academic Press, (1988).

[For89] Forcellese, G., Temperini, M., A system for automated deduction based on the Connection Method, Rapp. IASI 268, (1989).

[For90] Forcellese, G., Temperini, M., Towards a logic language: an object-oriented implementation of the Connection Method, LNCS 429, Springer Verlag, (1990).

[Foc90] Forcheri, P., Molfino, M.T., Educational software suitable for learning programming methodologies, Proc. CATS'90, 161-164, (1990).

[Gal86] Gallier, J.H., Logic for Computer Science, Harper & Row Publishers, (1986).

[Ing78] Ingalls, D., The Smalltalk 76 Programming System Design and Implementation, Proc. 5th POPL Conf., 9-16, (1978).

[Kow83] Kowalski, R., Logic for Problem Solving, The Computer Science Library, (1983).

[Kre90] Kreczmar, A., et al., LOGLAN '88 - Report on the Programming Language, LNCS 414, Springer Verlag, (1990).

[Lim90] Limongelli, C., et al., Abstract Specification of Mathematical Structures and Methods, LNCS, 429:61-70, Springer Verlag, (1990).

[Lim91] Limongelli, C., Temperini, M., Abstract Specification of Structures and Methods in Symbolic Mathematical Computation, Theoretical Computer Science, to appear, (1991).

[Mey88] Meyer, B., Object Oriented Software Constructions, Prentice Hall, (1988).

[Mio88a] Miola, A., Limongelli, C., The amalgamation of numeric and algebraic computations in a single integrated computing environment, Rapp. DIS 18.88, (1988).

[Miob88] Miola, A., Mora, T., Constructive lifting in graded structures: a unified view of Buchberger and Hensel methods, J. Symb. Comp., 6: 2/3, Academic Press, (1988).

[Mio90a] Miola, A., (ed.), Computing tools for scientific problems solving, Academic Press, (1990).

[Mio90b] Miola, A., (Ed.), Design and Implementation of Symbolic Computation Systems, LNCS 429, Springer Verlag, (1990).

[Mio91] Miola, A., Symbolic Computation Systems, In: Encyclopedia of Computer Science and Technology, (Kent, A., Williams, J.G., eds.), Marcel Dekker, to appear, (1991).

[Mir87] Mirkowska, G., Salwicki, A., Algorithmic logic, PWN Warszawa & D. Reidel Publishing Company, (1987).

[Par91] Parisi Presicce, F., Pierantonio, A., An algebraic view of inheritance and subtyping, Proc. ESEC, (1991).

[Reg90a] Regio, M., Temperini, M., A short review on Object Oriented Programming: Methodology and language implementations, Proc. ACM Sympos. on Personal and Small Computers, (1990).

[Reg90b] Regio, M., Temperini, M., Object-Oriented Methodology for the Specification and the Treatment of Mathematical Objects, in "Computer Systems and Applications", E. Balagurusamy and B. Sushila (eds.), Tata McGraw-Hill, (1990).

[Reg90c] Regio, M., Temperini, M., Type redefinition and polymorphism in Object-Oriented Languages, Proc. TOOLS PACIFIC 90, (1990).

[Str86] Stroustrup, B., An Overview of C++, ACM SIGPLAN Notices, 21:11:7-18, (1986).

[Str87] Stroustrup, B., What is Object-Oriented Programming, ECOOP 87, (1987).

[Yun80] Yun, D.Y.Y., Stoutemyer, R.D., Symbolic Mathematical Computation. In: Encyclopedia of Computer Science and Technology, Vol.15, (Belzer, J., Holzman, A.G., Kent, A., eds.), Marcel Dekker, 235-310, (1980).

[Wan90] Wang, D., Some examples for testing an integrated system, Italian Nat. Res. Council, TASSO Report, (1990).

Parallel ELLPACK: A Development and Problem Solving Environment for High Performance Computing Machines

E.N. Houstis and J.R. Rice
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907

June 13, 1991

Abstract

In this paper we describe the design objectives and architecture of a development environment and methodology for building large scale computational models on parallel machines and its use for model/design prototyping. The aim of this system is to provide a machine independent problem solving environment that automatically maps the underlying computation to the specified targeting architecture, while it supports the endeavors of researchers. The system currently supports the numerical simulation of field problems on general purpose sequential machines, NCUBE and INTEL hypercubes, and Sequent shared memory parallel machines.

1 Introduction

Parallel (//)ELLPACK is a research project, whose main interests are a) the construction of a development environment and methodology for easily building parallel algorithms for the numerical simulation of various physical objects, and b) a machine independent problem solving environment (PSE) to support the prototyping (analysis, design) of physical objects. //ELLPACK system is a realization of these objectives which is upward compatible with the well known ELLPACK system [Rice 85]. Although the applicability of the system is extended beyond second order elliptic partial differential equations (PDEs), we kept the same acronym for identification and historical reasons. The current version of //ELLPACK supports the solution of nonlinear field problems [Weer 91] on NCUBE, Intel hypercubes, and Sequent shared memory machines. Efforts are underway to extend it to the CEDAR and GENESIS (an ESPRIT supported project) architectures. The system consists of three main subsystems that support the specifications of a computational model, its processing and the visualization of computed data. The development environment consists of a set of powerful linear and distributed data structures, a set of interfaces among well defined processes (called throughout modules or frameworks), and a number of tools supported by an appropriate algorithmic infrastructure for the automation of certain preprocessing or postprocessing operations. Figure 1 provides a view of the components of the system. It indicates the various components of the system used to build a computational model.

The precise design objectives of //ELLPACK architecture are described in Section 2. The features of the development environment and the intermediate language used for model/design specification are presented in Section 3. Finally, in Section 4 we give a brief description of the various tools and describe the editors that can be employed to compose the //ELLPACK program and specify its parameters.

2 Design Objectives

Parallel ELLPACK is a programming and problem solving environment for the specification and processing of PDEs on sequential and parallel machines. The main objective is to have a machine independent environment that attempts to reduce and hide the overhead due to parallel processing, while it supports the endeavors of a researcher. Another important aim of this system is to support some form of modular programming or programming in the large, allowing the development of new applications out of existing modules or frameworks. These modules or frameworks are characterized by fixed interfaces which can be used to support intermodule or interframe communication. By the term framework we mean generic processes (e.g., finite element or finite differences), which can implement different techniques by providing only the definition of some primitive element (i.e., finite element or finite difference stencil). One fundamental piece of information in the specification and simulation of physical objects is its geometry and topology. It is used to define a number of other data structures needed for the simulation. It has been observed [Chri 89] that geometric methods can be used to formulate new methods capable of exploiting the computational power of some type of parallel machines. This is usually achieved by splitting the geometric data or mesh data in some "optimal" way [Chri 90a]. So another design objective of parallel ELLPACK is for the programming environment to support the geometry decomposition methods. Although ELLPACK is an acronym associated with elliptic PDEs, the new system is evolving into one that supports the processing of general PDEs or predefined computational models. The name is carried out for historical and identification reasons. The ultimate objective of the parallel ELLPACK environment is to support the electronic prototyping of physical objects. The conceptual view of an electronic prototyping process for the design and analysis is shown in Figure 2. It consists of an initial design of the artifact, modeling the physical phenomena and an optimization facility, all interacting to produce a constraint satisfying design. The new parallel ELLPACK PSE architecture is extended to be able to accommodate the above process.

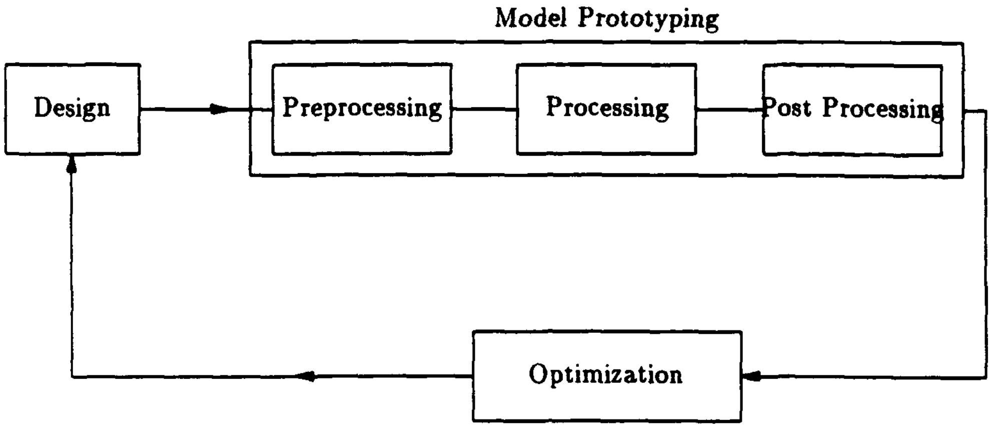


FIGURE 2: The electronic prototyping process.

3 Development Environment and Methodology

One of the objectives of //ELLPACK is to provide a programming environment for collecting, combining and evolving well defined PDE and system modules. In order to support the development of PDE based applications and solvers on sequential and parallel machines, one needs a set of powerful data structures that can store the information needed or generated in the various phases of the numerical simulation process. In the case of parallel processing some distributed form of these data structures must be provided. To support programming in the large or modular programming or the object oriented programming paradigm, we have extended the PDE oriented specification language ELLPACK [Rice 85], [Para 91]. These data structures and extensions are described below.

3.1 Data structures

It has been recognized [Finn 89] that geometric data structures play a fundamental role in any electronic design system. In parallel ELLPACK these data structures are used not only to specify the model geometry/topology, its attributes and meshes, but also to map the underlying computation to parallel machines. Figure 3 shows the relationship among these data structures. In this architecture the geometric model data serve as the core data to which all other data is associated. //ELLPACK provides a number of interactive tools for defining, displaying and manipulating these data structures. For two dimensional regions a textual and graphical parametric representation of the boundary is used. Figure 4 displays these two forms of model data. In three dimensions we are exploring the use of various existing geometric modeling systems, including a nonmanifold representation [Finn 89] and the PROTOSOLID approach [Vane 89].

We are developing two kernel models for the parallel implementation of finite differences and finite element methods in MIMD architectures. They use the geometry and mesh decomposition

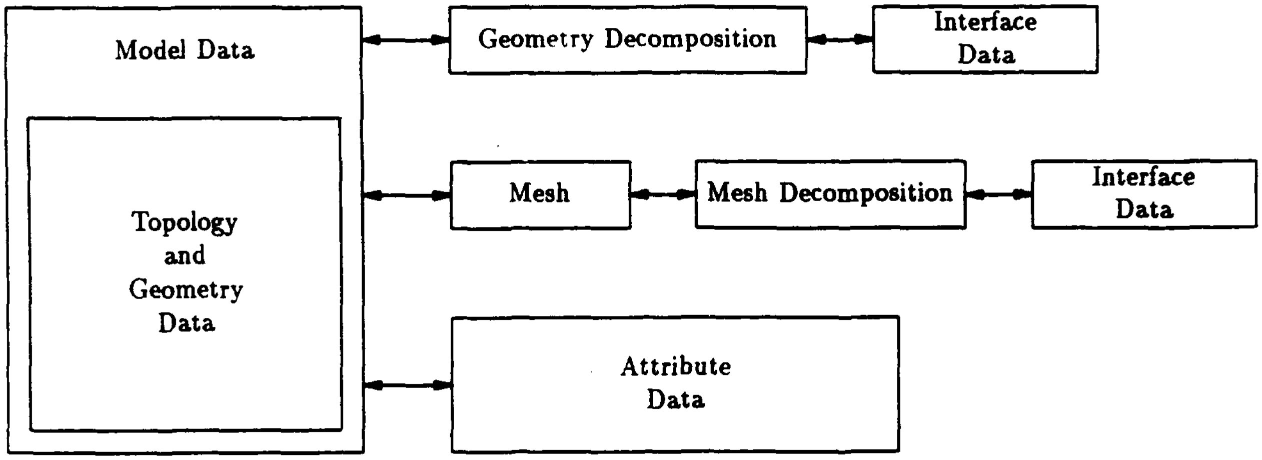
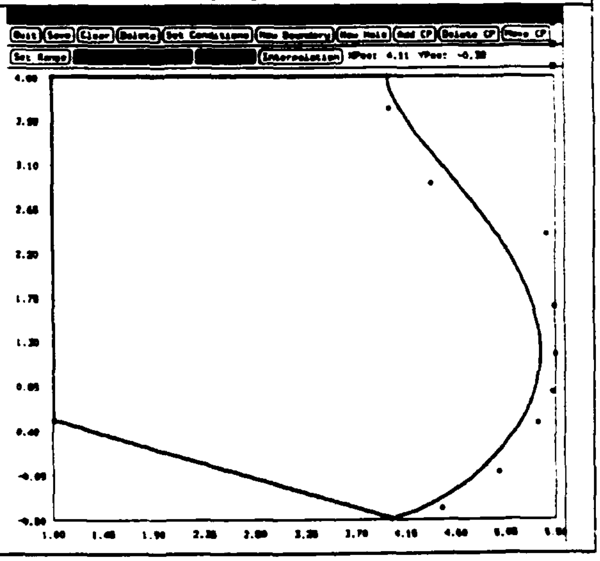


FIGURE 3: //ELLPACK geometry data specifications.

Boundary Segment	Bondary Specification Tool
see listing below	

```
boundary.
    U = true(x,y) on x = 4 + .1 * P * (P-4.5)**2,&
                     y = -.5 + P                  &
                  for P = 0. TO 4.5
    U = true(x,y) on x= 5 - P,                    &
                     y = 4.0                      &
                  for P = 1. TO 4.
    U = true(x,y) on x= 1.,                       &
                     y = 4.5 - P                  &
                  for P = .5 TO 4.
    U = true(x,y) on x= 1. + 3*P,                 &
                     y = .5 - P                   &
                  for P = 0. TO 1.
```

FIGURE 4: //ELLPACK model data specifications.

data to generate and store, in distributed sparse data structures, the algebraic data that constitute the computational model of the simulated objects. The geometry and mesh decomposition data are a set of hierarchical data structures that completely define the subregions, substructures or subdomains, together with the actual boundaries and interfaces to neighboring subdomains [Chri 89], [Chri 90], [Chri 90b]. The generation of this set of data is based on finite difference or finite element discretizations of the geometry. The relationship between the algebraic and solution data to the decomposed data is shown in Figure 5. Although //ELLPACK provides explicit support for geometry decomposition or splitting methods, one can also formulate and implement purely matrix partitioning techniques on the discrete distributed data.

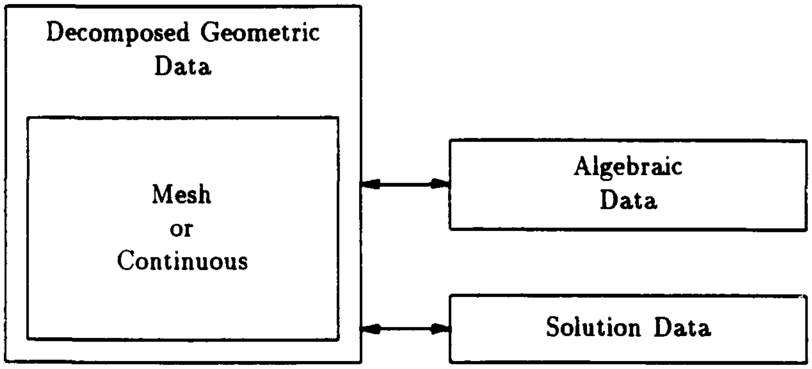


FIGURE 5: The relation between decomposed geometric data and algebraic data.

To support modular programming for ELLPACK [Rice 85] we have identified a number of interfaces corresponding to some specific frameworks (discretization, indexing, solution, output). The user can use these interfaces to develop other instances of such frameworks (or modules) and easily integrate them. We are currently designing a tool which will allow the visualization of algebraic data structures and interfaces. The //ELLPACK environment supports the automatic generation of most of these data structures and its algorithmic/software infrastructure can be extended easily. ELLPACK, and consequently //ELLPACK, has been designed not only to allow model prototyping, but to support many *research* endeavors related to development of new PDE solving software/algorithmic technologies.

3.2 Model specification and module-composition language

To support the specification of the PDE problem and the composition of PDE solving modules, we use a PDE language which is an extension of ELLPACK language [Rice 85]. The parallel ELLPACK language is upward compatible with the sequential ELLPACK and it will be implemented in C^{++}. It provides a single, machine independent PDE language capable of generating the control program of a specified problem for a variety of machines. In the following, we describe these extensions and indicate the ones that already have been implemented. To indicate the target machines its characteristics and configuration, we added the *machine segment* and extended the definition of other segments to indicate their execution with respect to the configuration (e.g., host or node) of the parallel machine. Some machines require that extensions (e.g., NCUBE 1). The ELLPACK applicability is to be expanded to accommodate PDE problems that can be described by a set of equations of the form

$$Fu \equiv F(t, x, y, z, D_t^i u,\ D_x^\kappa D_y^\ell D_z^m u, \int_{\partial\Omega} D_x^\kappa D_x^\ell D_z^m ds) = 0$$

where

$$u \equiv (u_1, \ldots, u_n)^T \text{ and } Fu \equiv ((Fu)_1, \ldots, (Fu)_n)^T$$

are defined in some region $\Omega \subset \mathbf{R}^\alpha$ with boundary $\partial\Omega$. Appropriate boundary/initial conditions are specified on the boundary of Ω. Currently, a subset of such PDEs can be accommodated by applying automatically some symbolic preprocessing [Weer 91] and a mixed FORTRAN/ELLPACK code. The ability of specifying two or more interactive models over subregions of Ω or interfacing regions is addressed in the new language. Although the PDE specification of certain numerical simulations is very common in applied mathematics, this is not true in certain engineering areas (e.g., structural analysis). Instead, a predefined computational kernel (finite element code) is available and the user chooses certain problem properties by selecting an appropriate element or a combination of elements and thus specifies the physical attributes on the continuous or discrete geometric data. We plan to interface such "foreign" systems with //ELLPACK and use its intermediate language and man-machine interface described later to provide the input expected by these systems.

In case of single or multiple two dimensional regions we will adopt the same parametric representation currently used, but we use tag variables associated with parts of the problem region (interior, boundary, interfaces). These variables may be referenced later in the //ELLPACK program. For three dimensional regions, the textual specification is in general impractical. We plan to use the intermediate language of a high level CAM/CAD system to represent the 3-D region.

The automatic generation of grids and meshes is a necessary tool of any numerical simulation system. We have recently added an automatic finite element mesh generator and created a new segment to accommodate orthogonal or non-orthogonal meshes and to import or export mesh data. The decomposition of "continuous" or "discrete" data and their use to formulate parallel PDE solvers has proven to be a very effective way to speed up computations [Chri 90a]. It is unrealistic to expect users to manually generate optimally such decompositions (it is NP complete problem), but, nevertheless, we provide a syntax for such textually specified data. Moreover. //ELLPACK provides a higher level tool and automatic techniques for generating these data. The decomposition segment also allows users to export or import compatible data. For the composition and specification of PDE solvers, we use the ELLPACK segments. Finally, some new graphical output facilities have been added that allow user to see computed solutions from various views.

4 Problem Solving Environment

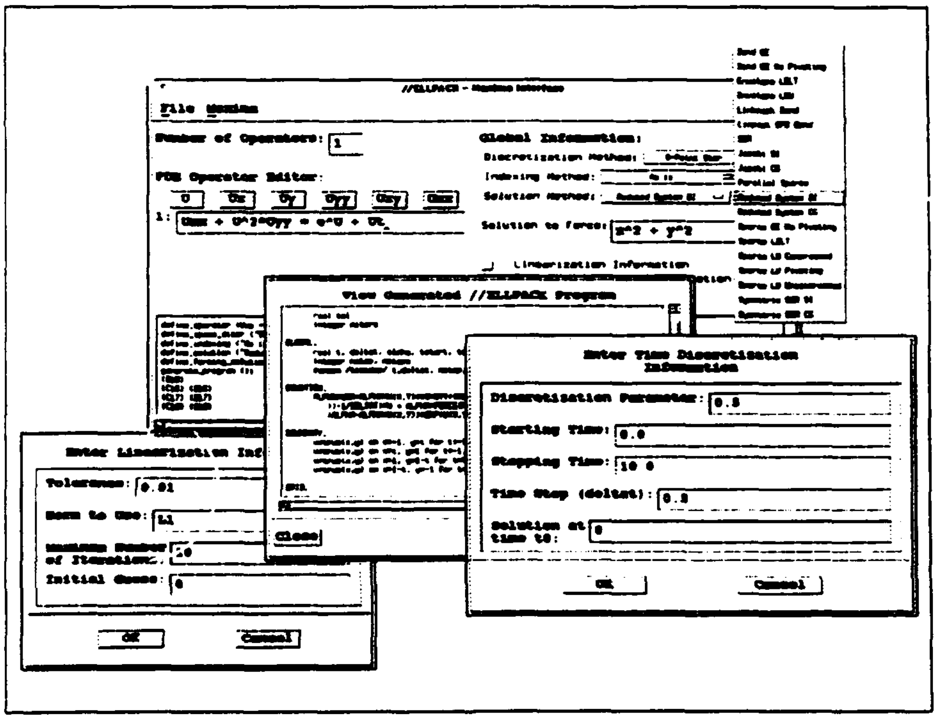
We have developed a number of communicating tools to support the symbolic transformation of PDE problems. These include the symbolic manipulation of input data, interactive graphical representation of two and three dimensional geometric regions, the graphical display and modification of meshes, the interactive definition or manipulation of domain decompositions, and the visualization of performance and solution data. These tools are implemented by X-windows based "intelligent" editors. These are described below along with examples of their user interfaces.

4.1 Processing subsystem

The //ELLPACK language described above allows the user to specify the input data in some "natural" form and explicitly state some of the computation/machine parameters. Often the problem specification is the result of many symbolic operations. Furthermore, the original problem might not be handled directly (e.g., it is nonlinear) by the underlying PDE solving infrastructure and so some derived information is required. Thus, we created a facility supported by an interactive editor and interfaced with MAXIMA that provides the user an environment to specify a PDE problem through a sequence of symbolic operations and transformations. Figure 6 depicts the user interface of this facility which is referred throughout as the *PDE specification editor*.

We have seen that the geometric and topological data of a model play a critical role in the numerical simulation process. The first step in the specification of some geometric object usually results in some analytic representation of the geometry and topology of the region. We have developed two X-window based interactive editors to support the design process of two and three dimensional domains. Figure 4b illustrates the 2D geometry specification tool.

The display and manipulation of grid and meshes is an important functionality of any PDE solving system. It allows one to adapt the numerical simulation to the behavior of the physical model. We have developed tools for orthogonal grids/meshes and unstructured finite element meshes for two and three dimensional meshes. Figures 7 and 8 display instances of the editor for two and three dimensional meshes. The decomposition of continuous or discrete domains is a very challenging problem from an algorithmic and implementation point of view. In //ELLPACK such decompositions play the role of steering mechanisms for the parallel processing of the computation. We have built a facility for obtaining such decompositions automatically. The display and manual manipulations of such decompositions and their mapping is possible through appropriate interactive editors. Figure 9 displays an instance of the domain decomposition editor.



//ELLPACK - Maxima Interface

Number of Operators: 1

PDE Operator Editor:

1: Uxx + U^2*Uyy = e^U + Ut

Global Information:

Discretization Method:

Indexing Method:

Solution Method:

Solution to Force: x^2 + y^2

Linearization Information

View Generated //ELLPACK Program

Enter Time Discretization Information

Discretization Parameter: 0.5

Starting Time: 0.0

Stopping Time: 10 0

Time Step (deltat): 0.5

Solution at time t0: 0

Enter Linearization Inf

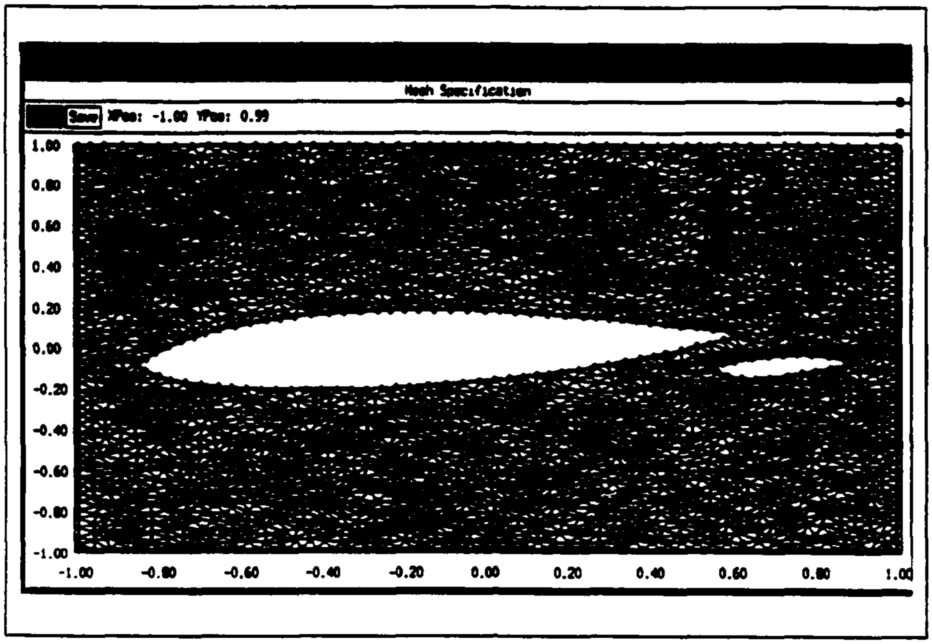
Tolerance: 0.01

Norm to Use: L1

Maximum Number of Iterations: 10

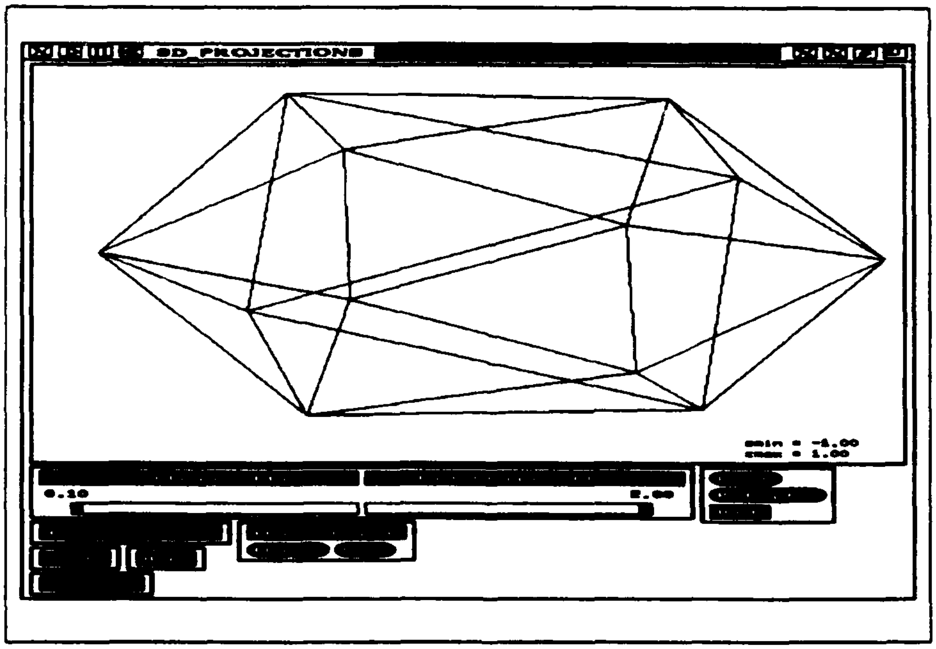
Initial Guess: 0

FIGURE 6: An instance of the PDE specification editor.



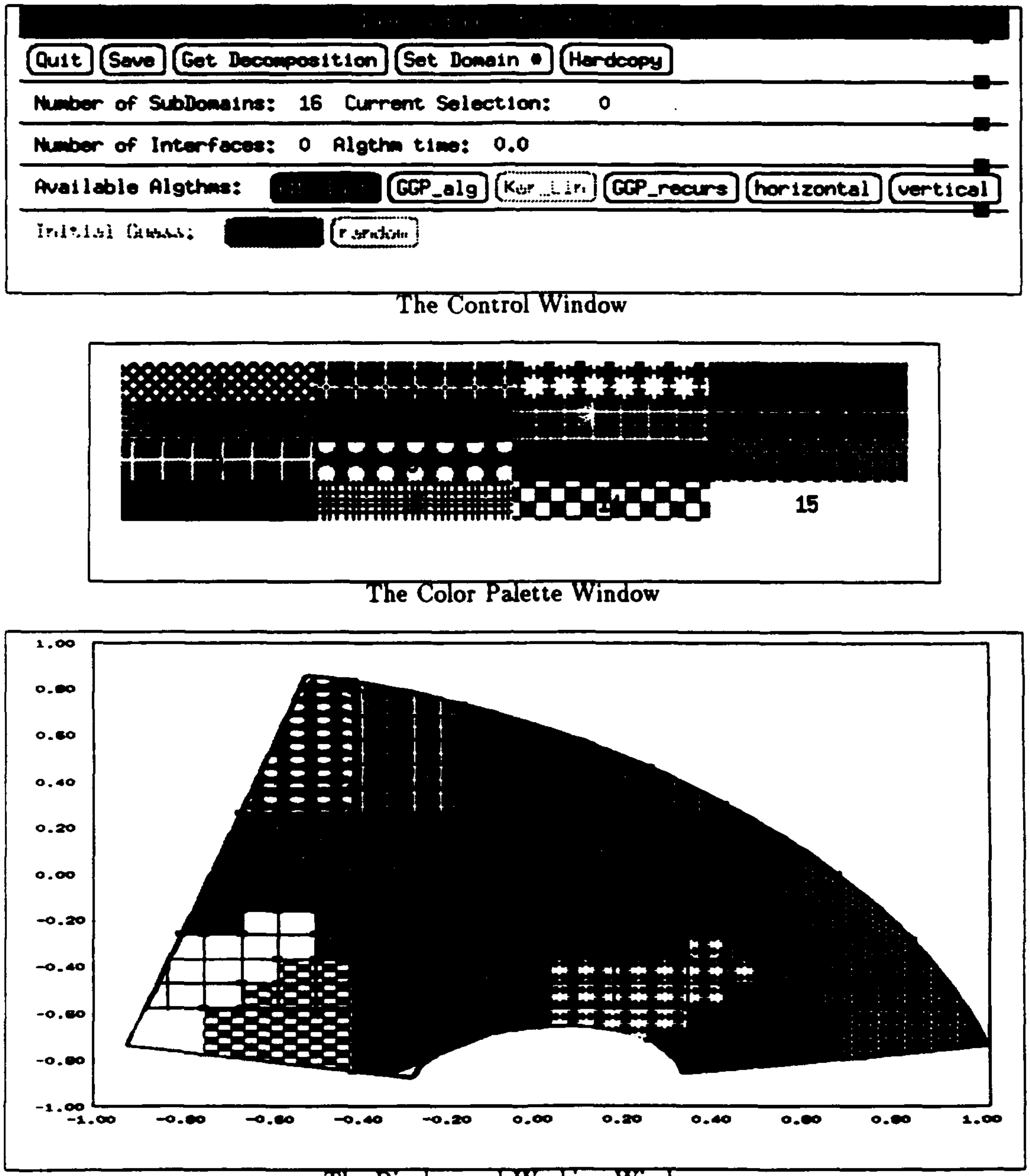
XPos: -1.00 YPos: 0.99

FIGURE 7: An instance of the two dimensional finite element mesh editor.



zmin = -1.00
zmax = 1.00

FIGURE 8: An instance of the three dimensional dinite element mesh editor.



Number of SubDomains: 16 Current Selection: 0

Number of Interfaces: 0 Algthm time: 0.0

Available Algthms:

Initial Guess:

The Control Window

The Color Palette Window

The Display and Working Window

FIGURE 9: An instance of the domain decomposition editor.

The appropriate selection of the grid or mesh size, its configuration and the choice of an efficient PDE solver for the given PDE problem, requires significant knowledge and experience. ELLPACK and its parallel extension provides a large library of PDE solvers, whose compatibility or effectiveness must be determined or assessed by the user. This open-ended algorithmic infrastructure needs the support of an "intelligent" front-end realized by an appropriate knowledge base system. We are building a such front-end for parallel ELLPACK. Figure 10 depicts an instance of its interface [Hous 91].

4.2 Pre-processing subsystem

The numerical simulation of field problems [Vemu 8] on MIMD machines is currently supported by a variety of solvers. Time-dependent, nonlinear and systems of PDEs are transformed automatically by the PDE specification tool and MAXIMA to create a sequence of linear elliptic PDE problems. For elliptic PDEs, we have developed a suite of finite element and difference parallel discretization modules. The solution of the corresponding algebraic systems are currently solved with a parallel version of ITPACK [Hous 89] or a sparse matrix solver [Mu 90], which we have developed for message passing machines. The development of band matrix methods based on the BLAS 2 and 3 is underway [Aboel 91]. One of the original design objectives of ELLPACK was the automatic collection of performance data and their storage in a database. This functionality is even more significant for //ELLPACK, since the amount of evaluation data is significantly larger

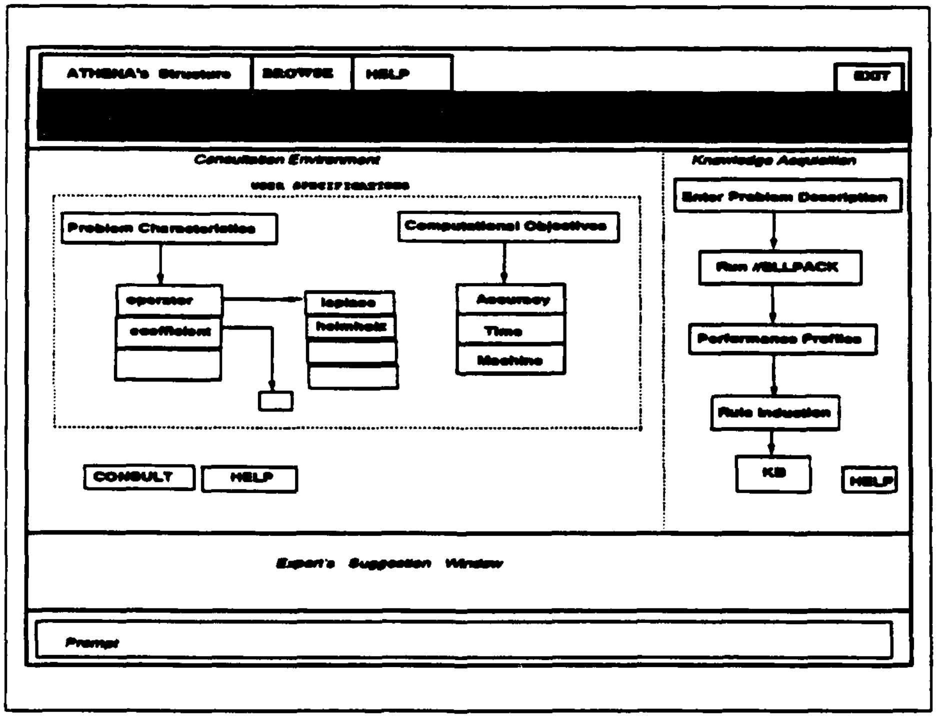


FIGURE 10: The interface of the "intelligent" front-end.

for parallel solvers. We have built a facility in //ELLPACK that collects and stores many performance indicators. Finally, an execution program tracing facility has been implemented that allows the user to monitor program behavior.

4.3 Post-processing subsystems

The aim of this subsystem is to support the visualization of solution and performance data, generate performance curves for various indicators and display them in a graphical form. The computed solution and its derivatives can be graphically displayed and rotated by 3-D graphics (see Figure 11). For the display of 3-D PDE problem solutions, the NCSA XDS facility is used [NCSA 89]. Two editors have been developed for the visualization of the evaluation data and the results of the analysis [Para 91], [Hous 91]. Instances of these editors are depicted in Figures 12 and 13.

Acknowledgements

The realization of this system is due to the contributions of the following research associates and students: Apostolos Hadjidimos, Catherine E. Houstis, Manolis Vavalis, Mo Mu, Panos Papachiou, Nicholas Chrisochoides, Sanjiva Weerawarana, Ko Yang Wang, Sang Bae Kim, T. Ku, Pelayia Varodoglu, Athanasios Gaitatzes, Meletis Samartzis, Stavros Kortesis, George Vanecek and William Bouma.

We thank Ko Yang Wang for providing the PS pictures and for proofreading the paper. The

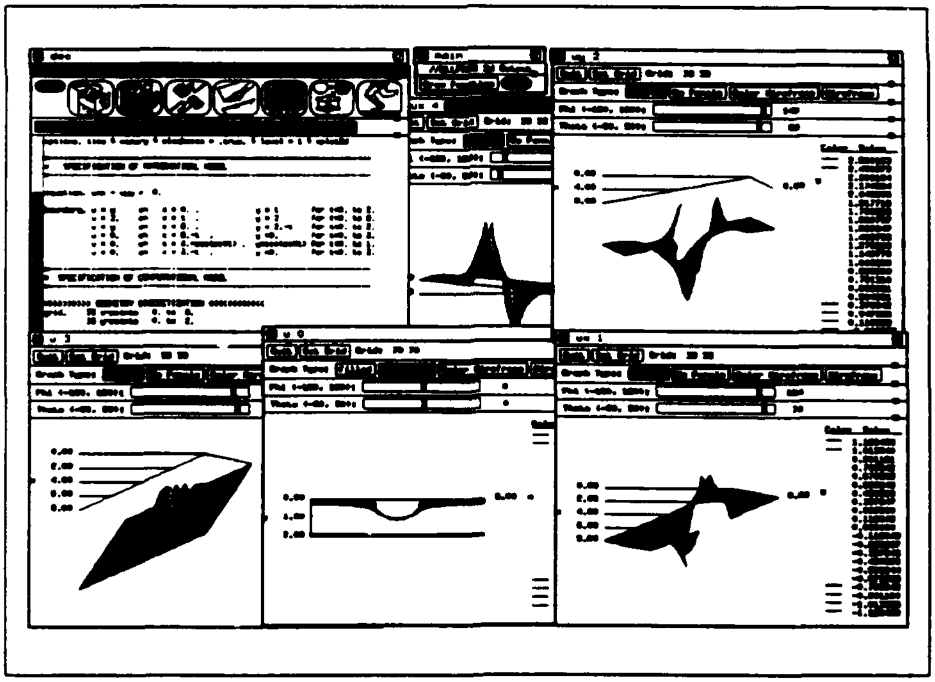


FIGURE 11: Editor for the visualization of solution data.

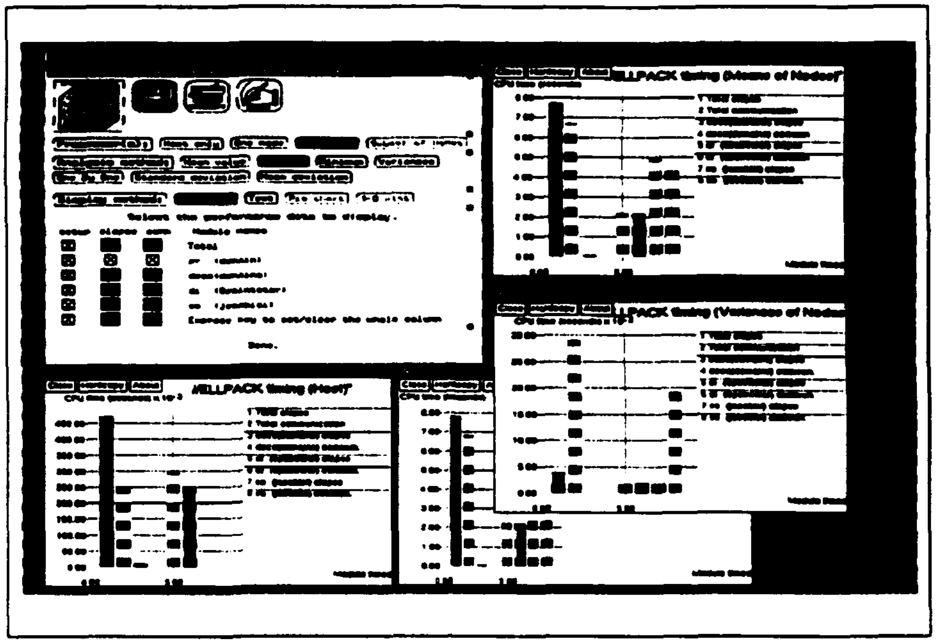

//ELLPACK timing (Host)
LLPACK timing (Means of Nodes)
LLPACK timing (Variances of Nodes)

FIGURE 12: Editor for the visualization of raw performance data.

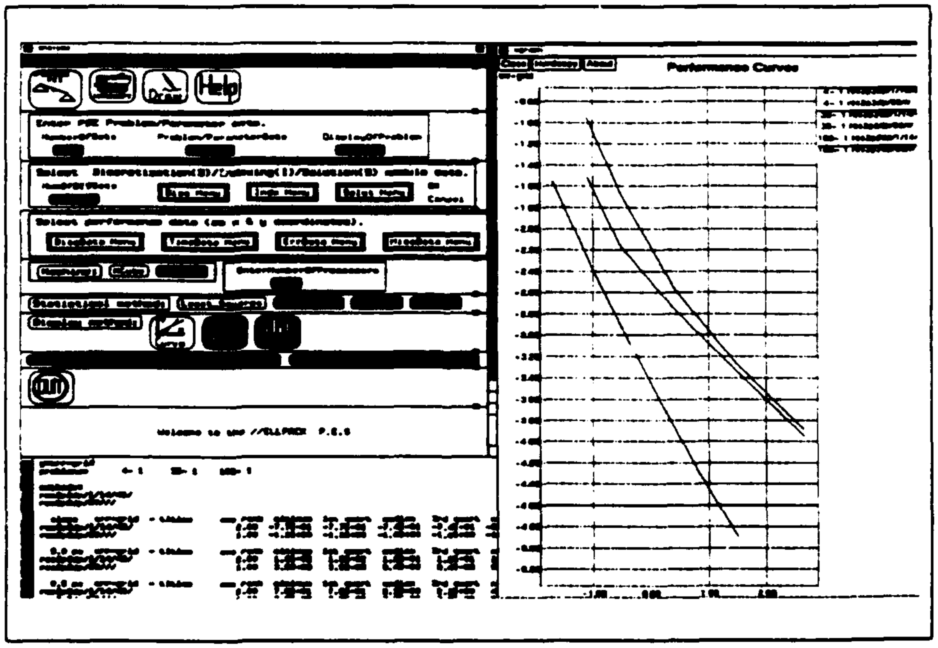


FIGURE 13: Editor for the analysis and display of performance curves.

above efforts were partially supported by the following grants: National Science Foundation CCR-8619817, AFOSR 88-0243, Strategic Defense Initiative (through the ARO) DAAL03-90-0107, ESPRIT project GENESIS.

References

[Aboe 91] M. Aboelaze, N.P. Chrisochoides, E.N. Houstis, and C.E. Houstis, *The parallel implementation of some BLAS 2 and 3 operations on distributed memory machines*, Proceedings of the Austria Parallel Processing Conference, September 1991, to appear.

[Chri 89] N.P. Chrisochoides, C.E. Houstis, E.N. Houstis, S.M. Kortesis, and J. Rice, *Automatic load balanced partitioning strategies for PDE computations*, in Int. Conf. on Supercomputing, 1989, pp. 99-107.

[Chri 90a] N.P. Chrisochoides, C.E. Houstis, E.N. Houstis, P.N. Papachiou, S.K. Kortesis, and J.R. Rice, *Domain decomposer: A software tool for mapping PDE computations to parallel architectures*, Domain Decomposition Methods for Differential Equations, SIAM, 1991, pp. 341-357.

[Chri 90b] N.P. Chrisochoides, E.N. Houstis, and C.E. Houstis, *Geometry based mapping strategies for PDE computations*, Proceedings of the International Conference on Supercomputing, Cologne-Germany, June 1991, to appear.

[Chrs 88] C.C. Christara, A. Hadjidimos, E.N. Houstis, and J.R. Rice, *Geometry decomposition based methods for solving elliptic PDEs*, Proceedings of Comp. Methods in Flow Analysis (H. Niki and M. Kawahara, eds.), Univ. of Okayana, 2, Japan, 1988, pp. 175-182.

[Finn 89] P.M. Finnigan, A. Kela, and J.E. Davis, *Geometry as a basis for finite element automation*, Engineering with Computers 5, 1989, pp. 177-160.

[Hadj 89] A. Hadjidimos, E.N. Houstis, M. Samartzis, J.R. Rice, and E.A. Vavalis, *Semi-iterative methods on distributed memory multiprocessor architectures*, Proceedings of the Third International Supercomputing Conference (E. Houstis and D. Gannon, eds.), ACM Press, 1989, pp. 82-90.

[Hous 89a] Houstis, E.N., T.S. Papatheodorou, and J.R. Rice, *Parallel ELLPACK: An expert system for the parallel processing of partial differential equations*. Math. Comp. Simul., **31**, 1989, pp. 497-508.

[Hous 89b] Houstis, E.N., J.R. Rice, N.P. Chrisochoides, H.C. Karathanasis, P.N. Papachiou, M.K.Samartzis, E.A. Vavalis, and K. Wang, *Parallel (//) ELLPACK PDE solving system* CAPO technical report CER-89-20, Purdue University, West Lafayette, IN, 1989.

[Hous 90] E.N. Houstis, J.R. Rice, N.P. Chrisochoides, H.C. Karathanasis, P.N. Papachiou, M.K. Samartzis, E.A. Vavalis, Ko Yang Wang, and S. Weerawarana, *Parallel ELLPACK: A numerical simulation environment for parallel MIMD machines*, Proceedings of the Fourth International Supercomputing Conference (J. Sopka, ed.), Amsterdam, 1990, pp. 96-107.

[Hous 91] Houstis, E.N., C.E. Houstis, J.R. Rice and P. Varodoglou, *ATHENA: An Knowledge Base System for //ELLPACK*, Proc. Conf. Symbolic-Numeric Data Analysis and Learning, 1991, to appear.

[Mu 90] Mo Mu and John R. Rice, *A New Organization of Sparse Gauss Elimination for Solving PDEs*, CAPO report CER 90-22, Department of Computer Science, Purdue University, July 1990.

[NCSA 89] NCSA group, *NCSA X Data Slice for the X Window System*, Technical report. Nat. Ctr. Supercomputer Appl., University of Illinois at Urbana-Champaign, Sept., 1989.

[Para 91] Parallel ELLPACK group, *Parallel ELLPACK Programming Environment: User's guide*, CAPO report, CSD-TR-91-034, CER-91-12, Department of Computer Science, Purdue University, April 1991, 58 pages.

[Rice 85] Rice, J.R. and R.F. Boisvert, *Solving Elliptic Problems Using ELLPACK*, Springer-Verlag, New York, 1985.

[Rice 88] Rice, J.R., *Supercomputing About Physical Objects*, Supercomputing, (eds, E. Houstis, T.S. Papatheodorou, C.D. Polychronopoulos), Springer Verlag, New York, 1988, pp. 443-455.

[Vane 89] Vanecek, G. Jr., *PROTOSOLID: An inside look*, CAPO report CER-89-26, Department of Computer Science, Purdue University, November 1989, 36 pages.

[Venn 81] V. Vennin and W.J. Karplus, *Digital computer treatment of partial differential equations*, Prentice-Hall series, 1981, 449 pages.

[Weer 91] S. Weerawarana, E.N. Houstis, and J.R. Rice, *An interactive symbolic-numeric interface to parallel ELLPACK for building general PDE solvers*, Proceedings of the Integration of Numeric and Symbolic Computing, Saratoga-Springs, June 1990, to appear.

DYNAMIC SOFTWARE RECONFIGURATION SUPPORTS SCIENTIFIC PROBLEM SOLVING ACTIVITIES

James M. Purtilo

Institute for Advanced Computer Studies *and*
Computer Science Department
University of Maryland
USA

This research is supported by the National Science Foundation under contract NSF CCR-9021222.

Folklore from the early days of our field has it that problem solvers, who employed numerical approximation techniques, could regularly halt their computer in order to manually inspect the progress of their program. Based upon what they saw, and knowing the layout of their program in memory, the user could even alter a few parameters — say, step sizes or tolerances — in order to adapt the algorithm's behavior.

Since those early days, we have evolved higher-level programming languages, more powerful computers, and better user interface software than ever before. But the level of dynamic control that a user has over his program is still at the level of halting, selective 'tweaking' of values, and resumption of execution. Only small-scale adaptations can be performed, since current software technology requires the program structure to remain essentially the same. But what if — dynamically — the user discovers that an alternate method altogether could take over computation and perform better? Even though the intermediate program state might be easily used by an alternate program component (e.g., integrators often use the same data structures but vary only in the order they update grid elements), the user cannot change methods unless he has explicitly designed this capability into his program right from the start.

Anticipating all possible alternatives that might arise in a long-running scientific computation is infeasible, and the cost of terminating the computation in order to restart it can be prohibitive. Ad hoc methods — such as inducing a core dump and manually transferring the core map to another program unit — are not to be trusted. A technique for dynamically reconfiguring software applications is needed, and, as we will show, several forms of reconfiguration are desirable for scientific computing.

Based upon such motivation, a technology for developing reconfigurable programs is now emerging. The purpose of this paper is to describe these software mechanisms, and then to relate them to potential applications in the scientific computing community.

1 RECONFIGURATION TECHNOLOGY

Capabilities for controlling changes to an application's implementation, even as it executes, are increasingly in demand. Users of highly-available systems must perform maintenance in place on software components; software developers may discover the need to transparently instrument some application only after it has been functioning for some long time; and, ultimately, both users and administrators alike may desire a facility for either relocating or reconfiguring a running application in order to improve its performance.

Techniques for static control of application programs have been available for years under the software engineering label configuration management. However, dynamic techniques, especially those that can be employed in heterogeneous systems, have not been so generally available. To date, loosely-coupled distributed applications have had the most success in dynamic alterations to components. But to consider only applications of this type would be to overlook a wide range of reconfiguration activities that may still be of great value to users. As described in [PuHo91], software can be reconfigured in three general ways: geometric, topological, and implementational.

GEOMETRIC CHANGES. By the "geometry" of a program we refer to how it is mapped onto the available computing system or systems. When the program is executed within a single process, then typically there are few geometric decisions to make: one simply decides where the process is to execute, and therefore the only form of geometric reconfiguration is to change where that process executes. This is also called process migration, and while it is relatively straight forward in homogeneous systems, heterogeneous process migration represents a serious challenge to the software systems community. When the application is itself a parallel or distributed program, then the user may choose to change where only some of the components execute. Thus, whereas the local association of component interfaces remains fixed, the geometry may be altered to reflect new demands, for purposes of load balancing, software fault tolerance, adaptation of communication patterns, and finally migration of tasks in order to access resources only available on other proce^s^sors. Alternately, the user may choose to change the medium of communication between executing processes.

TOPOLOGICAL CHANGES. Also referred to as "structural reconfiguration," this form of change to a program occurs when the interfaces between program components are rebound. Whereas geometric reconfiguration maintains the logical association between components — changing only where those components are executed — topological reconfiguration reflects the fact that new components may be introduced, old modules may be removed, and the association between the interfaces of any of these modules may change. For example, a user may choose to introduce into the application a package for solving non-linear systems, replacing another system solver, and then cause future calls to the solver to be directed to the new package.

IMPLEMENTATION CHANGES. In this case, individual components within the program may change, even though the overall topology and geometry remain the same. A user may request changes in subroutines or functions *in place*, rather than reconfigure based upon addition and deletion of other components, as was the case in topological reconfiguration. The primary distinclying host platforms. Libraries must be available to provide hooks into the communication systems, some stub generation facility may be needed, and link editing tools must be able to operate based upon designs expressed in the configuration language.

3. The system requires a way to invoke the executables, and establish any communication channels between components as are dictated by the design. Moreover, the system must ensure that the *only* communication between processes occurs over channels that are controlled by the reconfiguration system. Should unanticipated communication channels be overlooked during a geometric change, then system failure may result when a process sends data to the "old address" of its correspondents.

4. The communication system must support translation of primitive data representations, in order to function in the presence of heterogeneity.

5. The system needs a mechanism for controlling any of the three major forms of reconfiguration, which may of course include preemptive control over some processes within the configuration.

Our approach to meeting these requirements is to organize applications in terms of *software bus* abstractions, where the communication substrate is extended to support certain primitive operations necessary to effect the desired reconfiguration steps. Previously, we have reported on the basic 'toolbus features', along with our experimental notations [PuJa90, Purt90]. More recently, we have begun using those notations as a base for dynamic reconfiguration research {PuHo91}. For scientific computing environments, a key operation is the *state capture* primitive, as will be described. First, however, we summarize the general characteristics of software bus organization that make it suitable for use in reconfiguration research, leaving details to the previous papers on this topic:

1. Our current implementation of a bus-based system, called Polylith, provides a module interconnection language (MIL) for users to express their designs. This MIL is not tied to any particular programming language, but rather is intended to provide a uniform basis for integrating components from many different language domains.

2. Polylith provides a variety of packaging tools to assist users in preparing their programs to execute in this environment. The general philosophy is one of accommodating the needs of existing languages and systems, not prescribing a standard to which existing languages and programs must conform.

3. As a software bus system, Polylith already provides a mechanism for invoking processes and then establishing communication channels between them via the underlying hosts. As such, it already provides many of the facilities needed to experiment with ways to later change the established configurations.

4. As discussed in [Purt91], one of the primary features of a software bus system is that it provides a way to integrate programs having very different data representation properties.

> Parameters and data references will have their representation coerced during transmission through the communication substrate, all transparent to the programmer of the individual components. Therefore, it is natural to consider building upon this resource when attempting to find ways to change a configuration within a heterogeneous environment.

Since all communication in software bus systems is centralized (logically, though perhaps not physically) within the bus substrate, this provides us with a ready data structure to examine should we seek to subsequently change the configuration at run-time. This is precisely the approach we take in this research, where the current bus specification — that encapsulates communication and interconnection decisions — is extended to also provide promises concerning how a configuration can be changed during the course of the application's life.

In the basic Polylith system, packaging tools are free to access other components through the communication system via such straight-forward accessors as `read`'s and `write`'s (in effect, a message passing system). With our extensions, the programmer is also able to access the communication system (bus) in order to determine the current configuration; this information is returned in a suitable data structure. Once available to a process, this configuration data structure can be edited (through appropriate calls) in order to describe how the new configuration is to look. The revised data structure can be passed back to the bus via another system call, at which point a sequence of low-level reconfiguration operations will be invoked in order to install the new configuration from the old. These system calls are complemented by availability of suitable locking mechanisms so that the process can prevent changes to the system state that might invalidate its own :econfiguration actions.

The details of these system calls, and examples of what they look like, are left to other papers — what is essential here is that in general the application programmer needs only to focus on correct implementation of each component's functionality, and then on the configuration itself, expressed in the Polylith MIL. In the case that implementation changes are anticipated, then the programmer is also asked to write a function that will, when called, expose the internal component state information, so that it can be transmitted to a replacement component. Polylith is responsible for all other packaging, compiling and linking obligations.

This does make the point, however, that some components are built to focus only on the application requirements, and others are built to 'know' they are functioning within a bus-based environment may therefore direct some reconfiguration steps. We refer to this as an *external* approach, in that you have one program unit operating on the configuration of another at run-time. This is as opposed to an *internal* approach, where an application is responsible for managing its own configuration. Clearly, the former can be used to implement the latter. However, since the reverse is not necessarily true, then we have chosen an external approach for its flexibility. Currently, we centralize the responsibility for reconfiguration within tools specially built for the job, such as the Minion editor [Purt89].

STATE-CAPTURE OPERATIONS. The most difficult reconfiguration activity to organize is that of module implementation, where state information necessary to the correct functioning of a replacement process is cached in the old version of the process. Simple copying of data segments may be insufficient for this activity, as with today's compilers and code generators there is no well-defined, application-independent way for the data elements to be identified or related. Moreover, the presence of heterogeneity in underlying platforms requires that the data be abstracted during extraction, so that its' representation can be coerced according to the appropriate data type rules.

Abstractly, our state-capture mechanism loosely follows a standard method for transmitting instances of abstract data types (ADTs) in a network [HeLi82]. In this method, representation maps for the type are developed and added to the type's signature. When called, one map returns an externalized data structure suitable for use within the communication subsystem; the other accepts an externalized structure and establishes an appropriate internal representation.

We have found that this approach can be extended so that processes can be treated as instances of an appropriately chosen ADT. In one scenario, the process can itself choose to divulge its state information through a representation map (installed either by automatic techniques, or by the designer); once divulged, the state can be relocated by the communication subsystem, then used by the inverse representation map to parameterize the invocation of any other valid implementation of that same ADT (that is, another copy of the program can start up where the previous process left off.)

Another scenario presents a more difficult situation, in that often the decision to reconfigure is made external to the component. In this case, steps must be taken to also capture the *control state* of the process. Unlike ordinary data types, a 'process' ADT also has a thread of control associated with it, and this may result in changes to the data state that are not anticipated by the agent invoking the representation map. In short, the data may not be in a 'safe' or consistent state when accessed. Hence, the software organization must ensure that appropriate locking is done on data elements, and moreover that an image of the control state is made available to the new process.

Schemes for ensuring consistent state for reconfiguration can be expensive, and in general, one of our other research objectives is to determine for which classes of programs we may automatically determine reconfiguration maps that are inexpensive to use. However, currently we rely upon the designer to assist in identifying and installing the maps for their application so they may be accessed by our environment during a reconfiguration step [PuHo91].

The representation map can be as simple as a function that, when called, simply returns the values of global variables that are desirable to be passed to any new implementation of this program. Implicitly this scheme allows reconfiguration to occur at any time, without locking, and is sufficient when the computation can proceed reliably even when those data are in an intermediate state. For example, say the operation being updated is an PDE system solver; here the data primarily consists of the grid data values. Many techniques will still converge to a satisfactory solution if the reconfiguration occurs mid-iteration, leaving only part of the grid updated from the last step. Another example is suggested to us in [GoRa91], where reconfiguration of signal processing applications is discussed. In this case, the system may reasonably drop some intermediate samplings or values during the reconfiguration step, and still provide satisfactory service upon resumption of normal communication.

The above scheme is most desirable from a performance point of view, since it entails *no* run-time costs until the actual reconfiguration step is initiated. However, it is not always feasible. In another scenario, the designer may be forced to determine windows during which reconfiguration is safe to proceed, hence locking out reconfiguration during updates of some data values. Coordinating the safe states via the underlying host operating systems (from which reconfiguration requests would be issued) can incur some run-time costs; this is of course undesirable for scientific computing applications. The most undesirable manifestation of this scenario occurs when suitable synchronization between the process and the operating system is not feasible. In this case, the designer may be forced to checkpoint the process state at regular intervals against the possibility of a future reconfiguration step. Checkpointing, of course, incurs costs that are generally unacceptable for scientific computing. A taxonomy of approaches available to designers, along with the corresponding costs to be paid, is being prepared for an extended version of [PuHo91].

2 EXPLOITING RECONFIGURATION

The previous section surveyed possible types of dynamic reconfiguration, and also described a workbench we are building to experiment with software reconfiguration. We now summarize how this facility specifically benefits scientific computing activities. The scenarios below are either possible within our framework, or have in fact already been experimented with to varying degrees within our laboratory. In each case, however, the description is of work in progress, and we anticipate producing detailed reports on each of these efforts as the projects evolve. These reports will include manuals for use, since our goal is to widely distribute the software infrastructure in order to solicit feedback from the community.

In scientific computing environments, the goal is to place powerful tools at the disposal of problem solvers. In the current generation of mathematical support tools (such as Maple, Macsyma or Matlab), only single "monolithic" programs are employed — the user only interacts with the one tool. Should other resources be required to solve the problem, then a serious question of representation and interconnection arises. Current systems do not easily interoperate with other tools, even though conceptually the user could benefit from such leverage.

To combat this interconnection problem that increases with the scale of a problem (as measured either in terms of the number of mathematical objects to be managed or in the number of subproblems the user must coordinate), we have previously described the potential benefits of tools supporting "problem solving in the large" [Purt89]. In this work, a separate user interface is provided to help the user coordinate — and, most of all, manage — diverse sub-problem solving steps. A principle feature of this work is that it shows how to use languages for expressing problem solving "plans" to such a support system. Since a user may not easily predict what tools may be required to solve problem (that is, it is often difficult to plan so far ahead into a task), the system we developed to experiment with these ideas provided some aspects of dynamic reconfiguration. Specifically, this system (called Minion), allowed dynamic introduction of new tools, along with rebinding of interfaces between these tools. These are topological reconfigurations only.

But a user may choose to replace a particular tool during an interactive session, and moreover there may be valuable data within the old tool that must be transmitted to the new implementation, i.e., geometric change may be desirable as well. One example of this that has arisen is in a large scale problem solving plan where the user has a computer algebra tool configured in one of the nodes. This tool was relied upon for its symbolic integration capability, and the session proceeded. At one point, however, the user was curious concerning a particular result, and decided to change integrators in order to compare some values later in the computation. As configured now in Minion, the user needed only to identify the necessary "state" variables from the computation; transmit those data on the reconfiguration interface; and finally request, via the graphics editor in our system, invocation of the replacement algebra tool. (The state data would be supplied to the new tool automatically.) In this case, state variables consisted of certain constant values and dependency rules.

At this time, we use the Minion editor as the mechanism by which reconfiguration directions are related from the user to a running application. In the case that state representation maps can be established within the program without need of checkpointing, then the application can execute without any loss of performance as compared to how it would execute outside of the reconfiguration system's framework.

The computer algebra scenario described earlier also has a numerical counterpart. Even as many numerical approximation techniques feature 'adaptive' steps, to adjust the solution approach to conditions of the particular problem, our approach to reconfiguration provides a radical extension to the concept of adaptation. The entire method can be changed. For example, a general ODE integrator may initially be employed upon a problem for which little 'context' information is available to guide in the choice of special techniques. During execution, either the program itself or an observer may recognize properties that could be exploited (e.g., stiffness, oscillatory behavior for which there are special methods, or discontinuities for which certain heuristics might yield results).

Our experience in this area has been that the key difficulty in this change is adapting the representation. In so many of these numerical activities, the method is tightly coupled with a particular representation scheme, and therefore a change in method requires possibly significant alteration in the data space as well. Moreover, it is unlikely that the data's old and new forms will correspond directly, requiring some conditioning. Participation from the user is essential in this activity, though we can report that once the user has found a way to describe an internal structure 'abstractly' in our system, subsequent reconfigurations involving that program are in a position to benefit immediately. However, because the adaptation of data can involve a potentially substantial amount of computation, it is becoming clear to us that dynamic reconfiguration is not something that is likely to ever be suitable for frequent, fine-grain use. This seems to be directly analogous to the situation in process migration systems, where the benefit of geometric change rarely seems to outweigh the cost of effecting that change frequently.

Other ways to exploit our system are now easy to envision. Once the user has expressed an application in terms of our configuration notation, then there is a basis for introducing instrumentation programs into the code as it executes. For example, a numerical integrator could be set up to assume success in taking steps — dynamically a user could introduce the 'hooks' so that intermediate results are temporarily diverted to a system solver to check the progress of the solution. This capability would have comparable benefits to application debuggers. More importantly, however, is the way in which this capability opens up a new way to interface front-end workstations to remote supercomputers. The benefits of interactively developing and experimenting with mathematical programs on a personal machine are well accepted. But after trying a few 'easy' tests on the workstation, users regularly discover that a simple change in the test data can result in an entirely different order of magnitude in complexity of the computation needed to solve the problem. With existing systems, this requires that the user wait until patience is exhausted, then terminate the program, port it to the high-end machine, and begin it again. But in our system, a simple graphics command could relocate the process to the supercomputer for faster execution - in short, the user would only relocate tasks to other machines only as needed, and without extensive interruption to the problem solving train of thought.

3 CONCLUSION

We have described a general framework for dynamic software reconfiguration, focusing on the broad objectives and motivating our solution approach. A workbench for experimenting with reconfiguration is being developed, and we have described how this is being used for preliminary application within scientific computing. Much research remains to be done, both in the general framework and in its specific use within scientific computing environments. The key remaining questions are to more precisely quantify the cost of each form of change, and also to determine the classes of programs for which a representation map can be calculated by automatic analysis. However, the direction is promising, and we will continue our efforts to bring this emerging software technology into the scientific computing arena.

REFERENCES

[ArFi89] Artsy, Y., and R. Finkel. Designing a process migration facility: the Charlotte experience. **IEEE Computer**, vol. 22, no. 9, (September 1989), pp. 47-56.

[Cher88] Cheriton, D. The V distributed system. **Communications of the ACM**, vol. 31, (March 1988), pp. 314-333.

[EiGe84] Einarsson, B., and W. Gentlemen. Mixed language programming. **Software – Practice and Experience,** vol. 14, (1984), pp. 383-396.

[GoRa91] Gorlick, M., and R. Razouk. Using weaves for software construction and analysis. *Proceedings of Int'l Conference on Software Engineering,* (May 1991), pp. 23-34.

[HeLi82] Herlihy, M., and B. Liskov. A value transmission method for abstract data types. **ACM Transactions on Programming Languages and Systems**, vol. 2, (October 1982), pp. 527-551.

[KrMa90]J. Kramer, J. Magee. The Evolving Philosophers Problem: Dynamic Change Management. **IEEE Transactions on Software Engineering**, vol. 16, (1990), pp. 1293-1306.

[PuHo91]Purtilo, J., and C. Hofmeister. Dynamic reconfiguration of distributed programs. *Proceedings of Int'l Conference on Distributed Computing systems*, (May 1991), pp. 560-571.

[PuJa90] Purtilo, J., and P. Jalote. An environment for prototyping distributed applications. **Computer Languages**, vol. 16, (1991), pp. 197-207.

[Purt89] Purtilo, J. MINION: An environment to organize mathematical problem solving. *1989 Int'l Symposium on Symbolic and Algebraic Computation*, (July 1989), pp. 147-154.

[Purt90] Purtilo, J. The Polylith software bus. To appear, **ACM Transactions on Programming Languages and Systems**, (1990).

[ThHa91]Theimer, M. and B. Hayes. Heterogeneous process migration by recompilation. *Proceedings of Int'l Conference on Distributed Computing Systems*, (May 1991), pp. 18-25.

ACKNOWLEDGEMENT

I am grateful to Christine Hofmeister, a doctoral student at the University of Maryland, for hei on-going collaboration in this area, and especially for her efforts in the implementation of this reconfiguration workbench.

Visual PDEQSOL: A Visual and Interactive Environment for Numerical Simulation

by

Yukio Umetani*, Chisato Konno*, Tadashi Ohta**
*Central Research Laboratory
Hitachi Ltd., Kokubunji
Tokyo 185, Japan

**Hitachi VLSI Engineering Ltd.
Kodaira, Tokyo 187
Japan

1. Introduction

Past problem solvers for PDEs (Partial Differential Equations) have been developed and used under the batch or time-sharing environment using the host computer and CRT terminals (cf. [1]-[3], [5]). They generally take the form of high level programming languages with code generators, or sophisticated driver for subroutine libraries. And some of them exploit the high performance of supercomputer architecture. The present PDEQSOL (Partial Differential Equation Solver) is one of these solvers, which possesses both finite difference method (FDM) and finite element method (FEM) as automatic discretization facilities. It has been applied to real-world complex problems (cf. [6]-[8]).

The advent of high performance graphical workstation of recent years enables these solvers to step up to the next stage from programming languages (cf. [4], [10]). We have been also trying to extend current PDEQSOL to be workable on the high performance graphic workstation which is connected to the supercomputer. We named this new system "visual PDEQSOL". The objectives of visual PDEQSOL is to improve man-machine interface of PDEQSOL drastically and to support the total simulation procedure in consistent manner under the graphic environment (cf. [9]). We have just implemented the initial version of the prototype system.

In this paper, the overview of visual PDEQSOL, its man-machine interface and characteristic functions such as guidance for expansion of PDEs to numerical algorithms are explained using the implemented window examples.

2. Outline of Visual PDEQSOL

The aim of visual PDEQSOL is to realize the continuous and consistent support of total simulation process consisting of modeling, execution and result analysis through the visual and intelligent interface.

2.1 The system structure

The developed system structure of the visual PDEQSOL is shown in Fig. 1. It consists of five subsystems, where the present PDEQSOL acts just as the code generator subsystem. Other subsystems are as follows.

The model visualizer supports the interactive modeling process, by which users can input physical domain and PDEs directly into the windows.

The PDEQSOL debugger and the run-time diagnosis subsystem is a high-level execution debugger with visual interface, by which users can break into the simulation program and collect information such as values of variables and matrix to diagnose the meshing scheme and numerical algorithms. The result analyzer is a checking tool for numerical accuracy and error of the calculated result, using graphing function for numerical variables.

These subsystems are developed on the UNIX based standard platform, that is X-window system and OSF/Motif, so that the high portability of the system is realized and the appearance and behavior of interactive windows are standardized.

2.2 Man-machine interface

The interactive control panel of the visual PDEQSOL is shown in Fig. 2. Each button on the panel stands for information which constitutes simulation model and processes. Users can interact with system and input or get information of simulation model through these buttons.

The buttons are classified into three groups defined in the top-down manner. The first one is the physical model which expresses physical information of simulation model, such as physical shape (Draw Model), variables, dominating equations (PDE), boundary and initial conditions (B. Cond, I. Cond) and so on. The second one is the mathematical model which expresses mathematical information to carry out numerical simulation, such as meshing scheme for domain (Mesh), numerical algorithm (Algorithm), and so on. The third one is numerical model which expresses numerical operation and information, such as the execution of computation (Execution) and execution process (Exec. Log), calculated numerical result (Num. Result) and so on.

When a user pushes a button, the subwindows and input guidance templates appear, by which the user can confirm or refer to pre-defined information and add new information. For example, when the Draw Model button is pushed, then the subwindow for physical region and menus for defining new region appears, as shown in Fig. 3. Or, when the button is pushed, then the subwindow for dominating equation and the template for inputting new equation appear. The latter subwindow includes scroll bar of pre-defined variables and usable operators, as shown in Fig. 4.

3. The Characteristic Functions of the Model Visualizer

In this section, the functions of the model visualizer is described, because the model visualizer is one of the most characteristic parts of this system.

Model visualizer has the specific functions which guide users to build simulation model quickly and correctly referring to numerical analysis knowledge. In this section, two examples of them, that is the guidance for inputting boundary conditions and the guidance for constructing numerical algorithms, are explained.

3.1 Guidance for inputting suitable boundary conditions

This system guides users to input suitable and sufficient boundary conditions for each boundary region according to the nature of PDEs. This system extracts the suitable form of boundary conditions for each boundary parts from the PDEs information and the draw model, and guides the user to select and complete boundary conditions. The guidance proceeds in the following manner.

The system extracts the boundary parts of the region where the PDEs are defined. For two dimensional case, sides which belong to only one face are extracted.

The system decides the form of the suitable boundary condition for each part according to the PDEs information and the kind of condition. For the Neumann boundary condition, the flux terms of the PDEs are extracted as the left hand side of the condition equation. For the Dirichlet boundary condition, the target variable to give the condition is automatically decided for each equation. To decide the appropriate target variable is not a trivial matter for simultaneous PDEs.

The system displays the template showing the above information. Only a user has to do is to select the kind of conditions and input the right hand side of the condition equation for every selected boundary part.

An example of this guidance is shown in Fig. 5. At first, a user selects the objective equations on the BCOND template. Then, the left hand side of the boundary condition appears for both kinds of condition , and for each boundary part selected by the system. After the user selects the kind of condition and inputs the right hand side of the condition, the completed condition equation is displayed on the specified boundary part and the user can visually confirm it.

3.2 Guidance for expansion of PDEs to numerical algorithm

Users can input PDEs with time-dependency, simultaneity and non-linearity. The system guides users to construct numerical algorithms according to the characteristics of the PDEs. The guidance proceeds in the following manner.

The system automatically determines the characteristics of the inputed PDEs by parsing and checking the terms in the equations.

The system inquires users how to treat these characteristics. Inquiry is in the order of time-dependency, simultaneity and, non-linearity. In this inquiry, the latter choice is depends on and narrowed by the former choice. If there remains no latter choice, it is automatically skipped. So users can decide the algorithm with minimum choices. The choices are implicit method or explicit method for time-dependency, lumped method (cf. [7]) or successive method for simultaneity, and Newton-Raphson method or successive method for non-linearity.

Finally, if the matrix solution is needed according to the choices, the repertoire of solution is shown for user's choice.

The example of this guidance flow is shown in Fig. 6. The objective equation is a non-linear convection-diffusion equation. If the explicit method is chosen for time-dependency, there remains no choice for non-linearity. Thus no further guidance is done. In the case shown in Fig. 6, the implicit method is chosen for it. Then, the guidance template is prolonged and the menu of choice for non-linearity appears.

The algorithm expansion is realized from these choices as follows. Expansion also proceeds in the order of time-dependency, simultaneity, and non-linearity.

New variables needed to constitute the indicated algorithm are added. For example, the variables to store temporary value for time iteration, non-linear iteration and so on are added.

Equations are expanded according to the choice of method. The equations are broken into plural statements consisting of loop control statements, assignment statements, SOLVE statements (cf. [6], [8]) and so on.

Finally, given initial conditions and boundary conditions are modified to adapt to the above expansions.

The example of algorithm expansion is shown in Fig. 7. The statements expanded from the original equation appear in the scheme subwindow.

The mechanism of this expansion is shown in Fig. 8. This explains the expansion process for time dependency and non-linearity. In each step, new variables are added, and conditions are modified and equations are expanded according to the indicated choice of numerical algorithms. For the expansion of time-dependency, the original equation is expanded to statements (A), because the backward-Euler method is chosen. New variable DT is added and initial condition is modified. For the expansion of non-linearity, the equation in the obtained statement is expanded to statements (B), because the newton-raphson method is selected. New variable DN is added and boundary conditions are also expanded.

4. Concluding Remarks

We have designed visual man-machine interface for general PDE problems, and have developed the system which assists a user through total simulation procedure. This system has the following remarkable features.

- The original problem can be inputed directly into the windows.
- The system guide users to construct simulation model using numerical analysis knowledge.
- The system has high portability.
- The a~pearance and behavior of interactive windows are standardized. From our evaluation, the total simulation procedures are shortened to 1/10 to 1/30 compared to the conventional environment using FORTRAN.

Acknowledgement

We express our great thanks to Mr. Jim DeLaHunt for his initial design of visual PDEQSOL user interface during his stay at Hitachi Central Research Laboratory in 1988-1989.

References

[1] A.F. Cardenas and W.J. Karplus, PDEL - A Language for Partial Differential Equations, Comm. ACM, March, (1970), pp. 184-191

[2] J.R. Rice, ELLPACK - An Evolving Problem Solving Environment for Scientific Computing, Proc. IFIP TC2/WG2.5, (1985), pp. 233-245

[3] Willi Schonauer and Eric Schnepf, Software Considerations for Black Box Solver FIDISOL for Partial Differential Equations, ACM Trans. on Math. Soft., (1987), Vol. 13, No. 4, pp. 333-350

[4] P.W. Gaffney et al., NEXUS: Towards a Problem Solving Environment (PSE) for Scientific Computing, ACM SIGNUM Newsletter, (1986), Vol. 21, No. 3, pp. 13-24 [5] Grant O. Cook, Jr., ALPAL: A Tool for the Development for Large-Scale Simulation Codes, UCID-21482, Lawrence Livermore National Laboratory, 1988

[6] Y. Umetani et al., DEQSOL - A Numerical Simulation Language for Vector/Parallel Processors, Proc. IFIP TC2/WG2.5, (1985), pp. 147-164

[7] C. Konno et al., Advanced Implicit Solution Function and its Evaluation, Proc. Fall Joint Computer Conference, (1986), pp. 1026-1033

[8] C. Konno et al., Automatic Code Generation Method of DEQSOL, J. of Information Processing, (1987), Vol. 11, No. 1, pp. 15-21

[9] C. Konno et al., Interactive/visual DEQSOL: Interactive Creation, Debugging, Diagnosis and Visualization of Numerical Simulation, Mathematics and Computer in Simulation, 31, (1989), pp. 353-369

[10] Wayne R. Dyksen et al., Elliptic Expert: An Expert System for Elliptic Partial Differential Equations, Purdue University Technical Report CER-88-48, CSD-TR-840(1988)

List of Figures

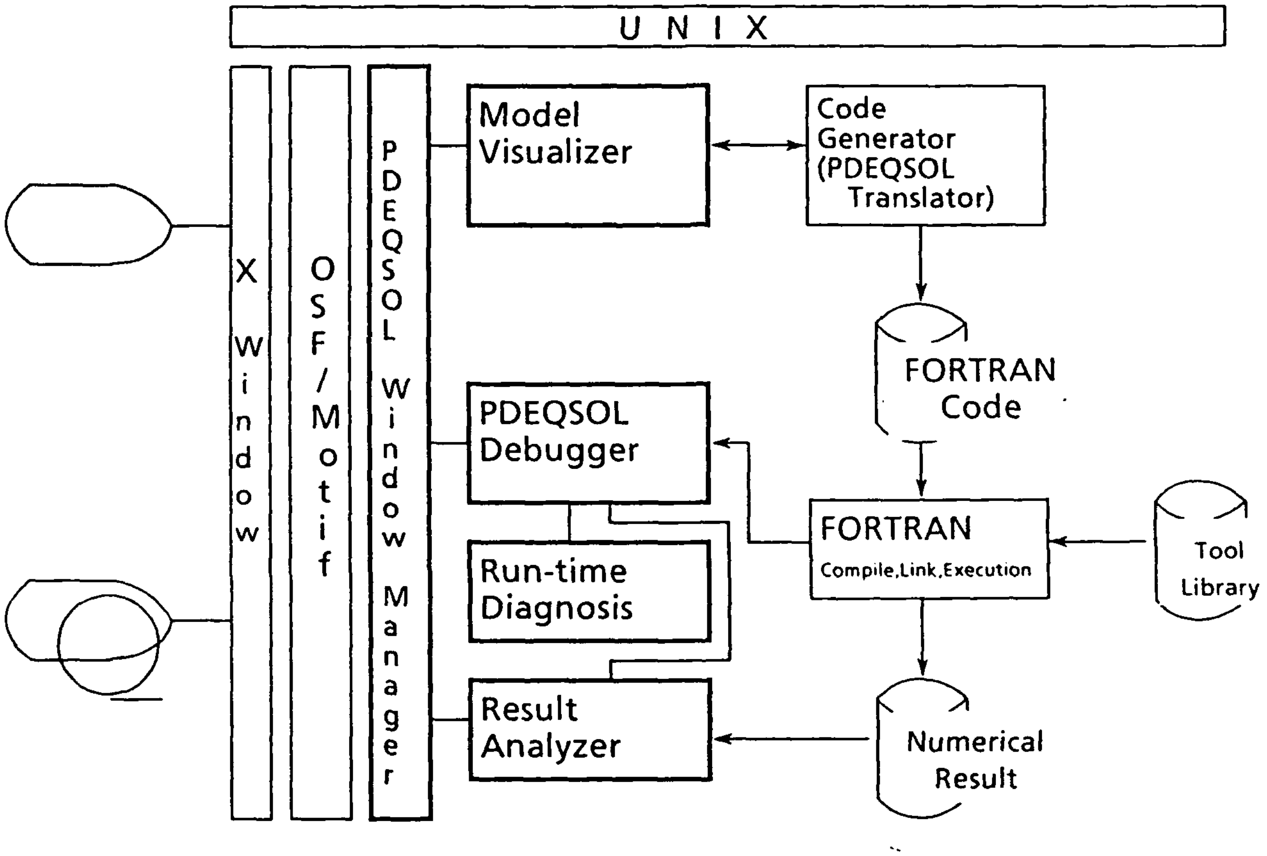
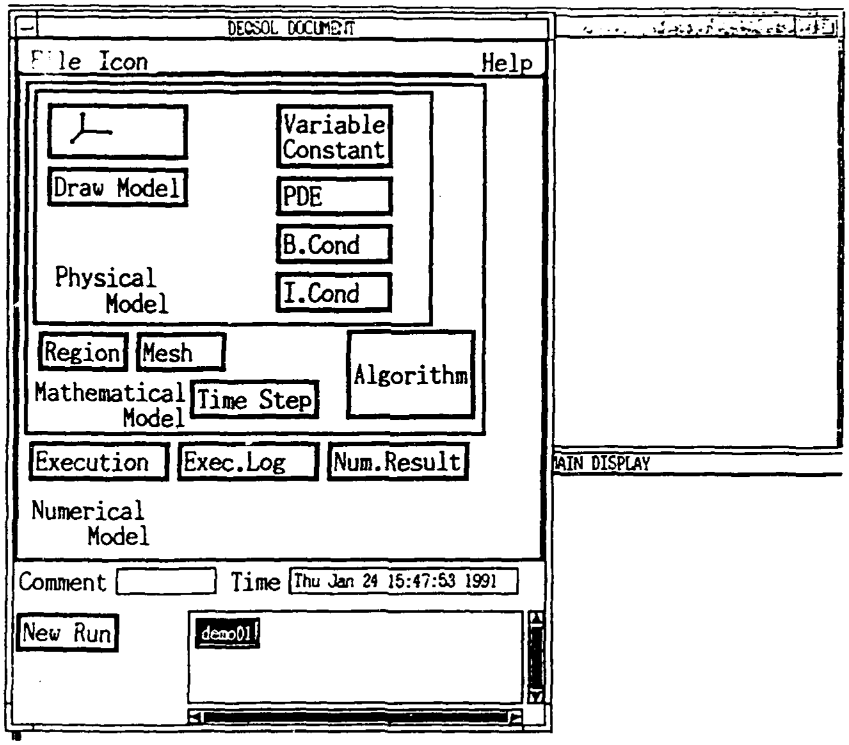


Fig.1. Visual PDEQSOL System



Time Thu Jan 24 15:47:53 1991

Fig.2 The Interactive Control Panel of the Visual PDEQSOL

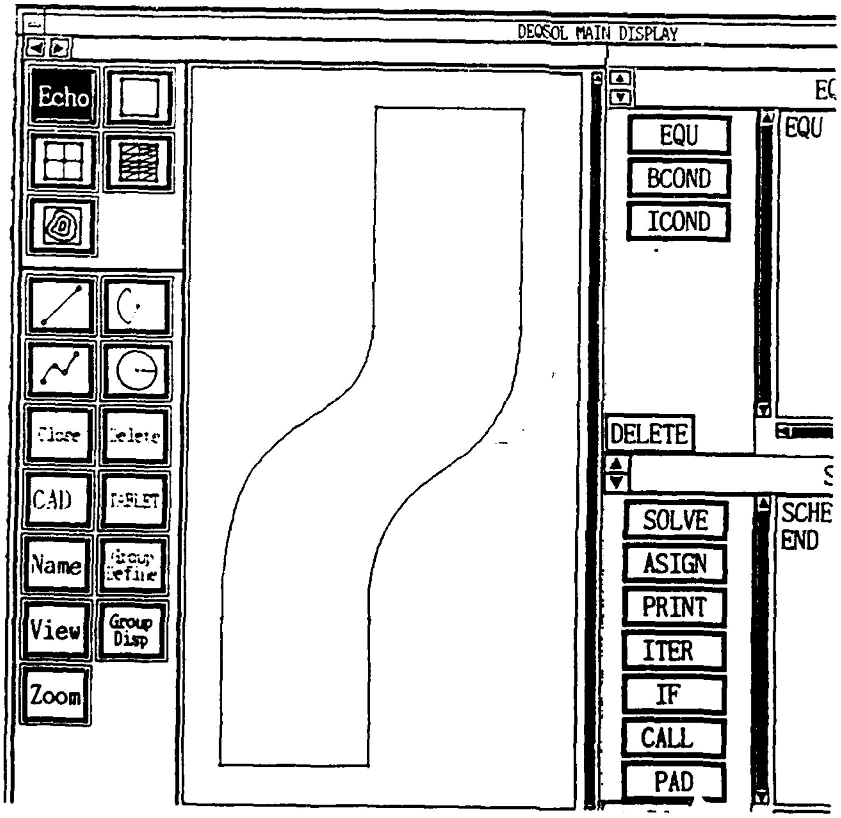
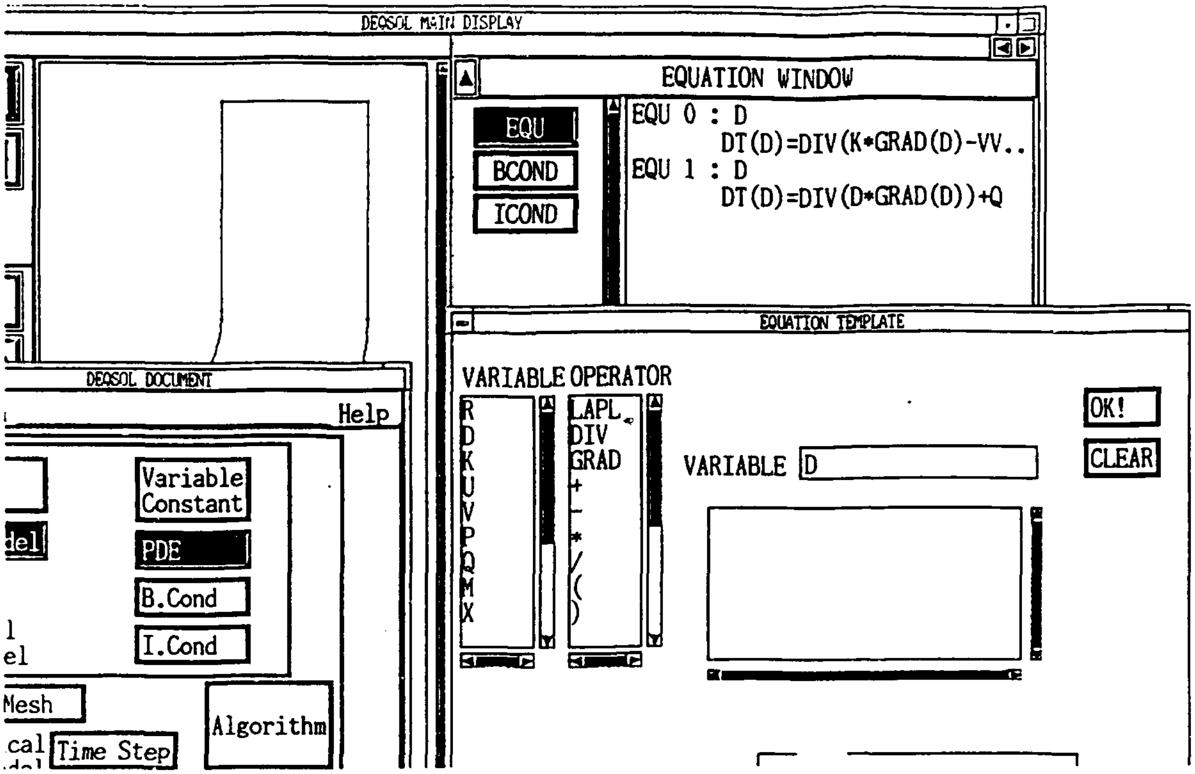


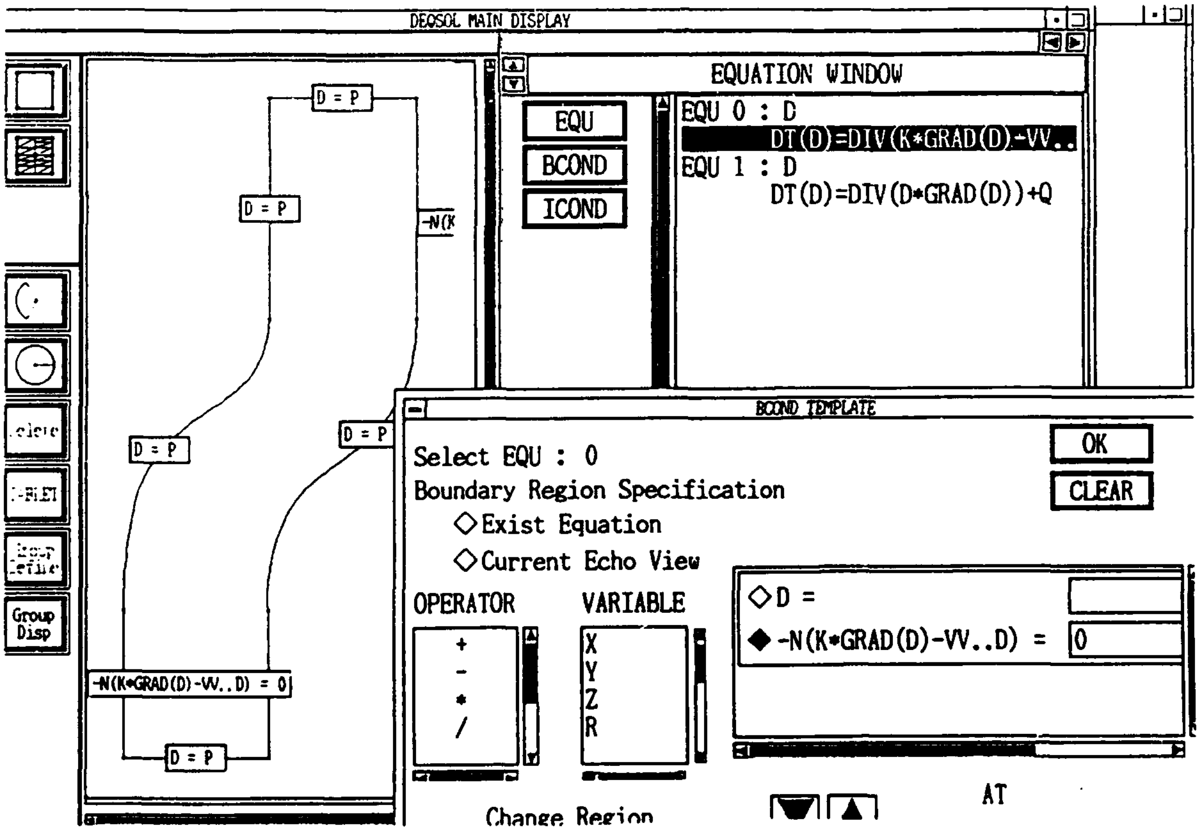
Fig.3 Example of the Interactive Window (Inputting Region)



EQU 0 : D
DT(D)=DIV(K*GRAD(D)-VV..
EQU 1 : D
DT(D)=DIV(D*GRAD(D))+Q

VARIABLE D

Fig.4 Example of the Interactive Window (Inputting Equation)



EQU 0 : D
DT(D)=DIV(K*GRAD(D)-VV..
EQU 1 : D
DT(D)=DIV(D*GRAD(D))+Q

Select EQU : 0

Boundary Region Specification

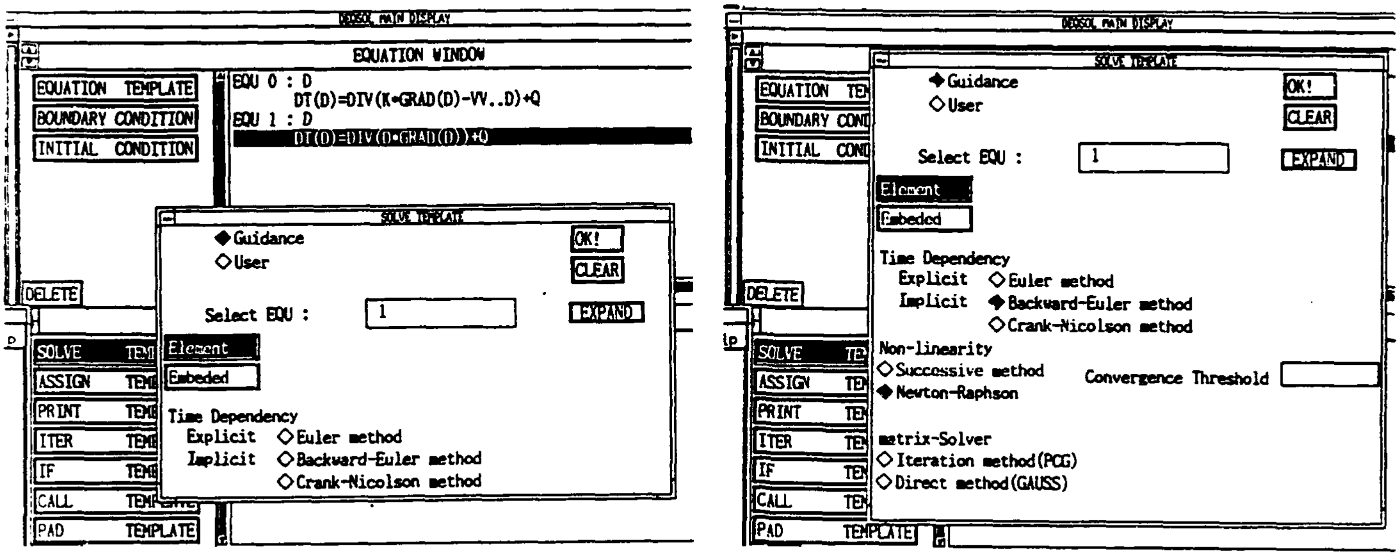
◇Exist Equation

◇Current Echo View

◇D =

◆-N(K*GRAD(D)-VV..D) = 0

Fig.5 Guidance Window of Boundary Conditions



```
EQU 0 : D
    DT(D)=DIV(K*GRAD(D)-VV..D)+Q
EQU 1 : D
    DT(D)=DIV(D*GRAD(D))+Q
```

◆ Guidance
◇ User

Select EQU : 1

Time Dependency
Explicit ◇ Euler method
Implicit ◇ Backward-Euler method
◇ Crank-Nicolson method

◆ Guidance
◇ User

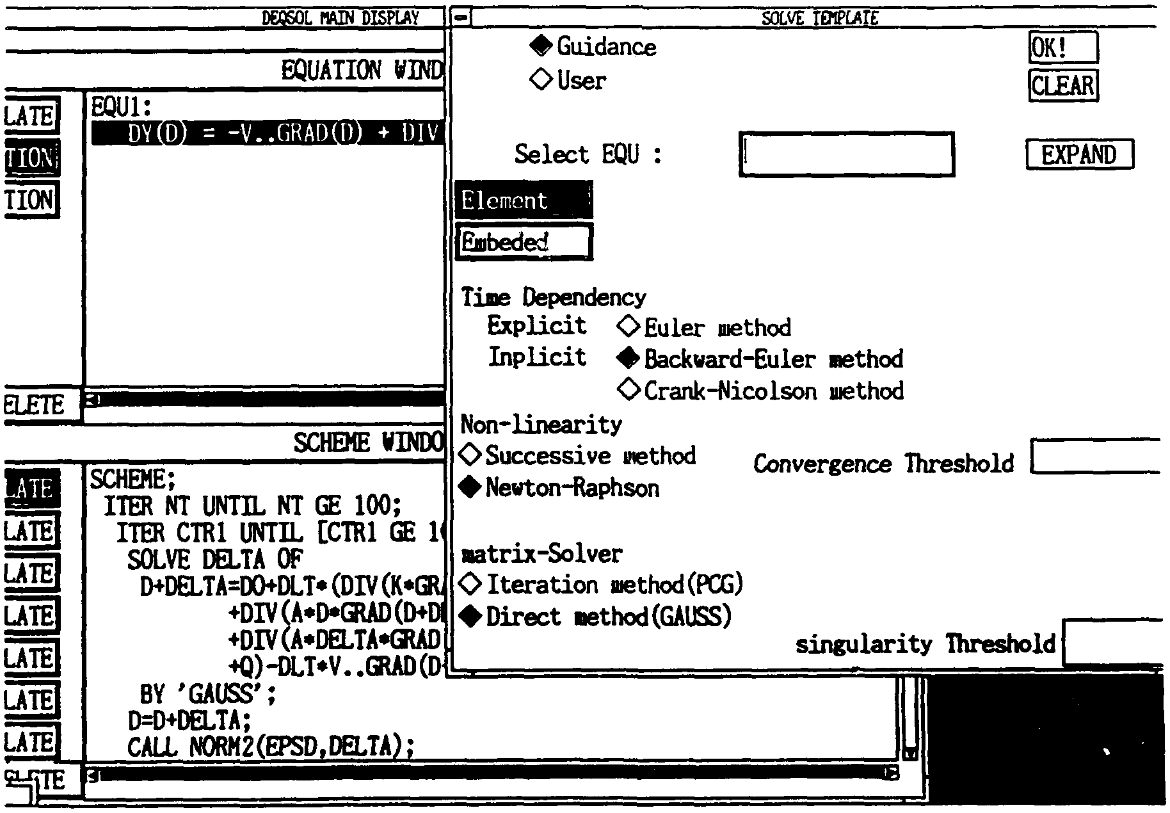
Select EQU : 1

Time Dependency
Explicit ◇ Euler method
Implicit ◆ Backward-Euler method
◇ Crank-Nicolson method

Non-linearity
◇ Successive method
◆ Newton-Raphson
Convergence Threshold

matrix-Solver
◇ Iteration method(PCG)
◇ Direct method(GAUSS)

Fig.6 Algorithm Expansion Guidance Window



```
EQU1:
  DY(D) = -V..GRAD(D) + DIV
```

```
SCHEME;
 ITER NT UNTIL NT GE 100;
  ITER CTR1 UNTIL [CTR1 GE 1
   SOLVE DELTA OF
    D+DELTA=DO+DLT*(DIV(K*GR
           +DIV(A*D*GRAD(D+D
           +DIV(A*DELTA*GRAD
           +Q)-DLT*V..GRAD(D
    BY 'GAUSS';
   D=D+DELTA;
   CALL NORM2(EPSD,DELTA);
```

◆ Guidance
◇ User

Select EQU :

Time Dependency
Explicit ◇ Euler method
Inplicit ◆ Backward-Euler method
◇ Crank-Nicolson method

Non-linearity
◇ Successive method
◆ Newton-Raphson
Convergence Threshold

matrix-Solver
◇ Iteration method(PCG)
◆ Direct method(GAUSS)
singularity Threshold

Fig.7 Example of Expanded Algorithm

```
VARIABLE   D;
EQUATION   DT(D)=DIV((K1+K2*D)*GRAD(D))+Q;
ALGORITHM  SOLVE D OF DT(D)=DIV((K1+K2*D)*GRAD(D)) + Q;      Ⓐ
BCOND      N..((K1+K2*D)*GRAD(D))=0  AT  WALL,
           D=A          AT  GRAND,
           ....   ;
ICOND      D=B  ;
              ↓
              ↓  Expansion of time dependency
              ↓
VARIABLE   D, DT;                                          (A)
ALGORITHM  ITER NT UNTIL (NT GE 100);                       ¦
             SOLVE D OF (D-DT)/DLT=DIV((K1+K2*D)*GRAD(D)) + Q; ¦    Ⓑ
             DT=D;                                          ¦
           END ITER;                                        ¦
BCOND      N..((K1+K2*D)*GRAD(D))=0  AT  WALL,
           D=A          AT  GRAND,
           ....   ;
ICOND      DT=B  ;
              ↓
              ↓  Expansion of non-linearity
              ↓
VARIABLE   D, DT, DN;
ALGORITHM  ITER NT UNTIL (NT GE 100);                                 (B)
             ITER NN UNTIL (NN GT 1000) OR (NDN LE eps);               ¦
               SOLVE DN OF (D+DN-DT)/DLT=DIV((K1+K2*D+K2*DN)*GRAD(D)) + ¦
                                          DIV((K1+K2*D)*GRAD(D) + Q     ¦
                     BY 'ILUBCG' ;                                      ¦
               D=D+DN;                                                  ¦
               CALL NORM2(NDN,DN);                                      ¦
             END ITER;                                                  ¦
             DT=D;
           END ITER;
BCOND      N..((K1+K2*D+K2*DN)*GRAD(D))+N..((K1+K2*D)*GRAD(DN))=0  AT  WALL,
           DN=A-D          AT  GRAND,
           ....    ;
```

Fig.8 Example of Algorithm Expansion

DISPLAY OF FUNCTIONS OF THREE SPACE VARIABLES AND TIME USING SHADED POLYGONS AND SOUND*

W. M. COUGHRAN, JR.† AND ERIC GROSSE†

Abstract. This talk will describe the visualization tools used in our scientific computing group to look at data and functions in two and three space variables. Emphasis is given to aspects that differ from the prevailing style elsewhere, and the points made will be illustrated with a videotape of representative example of the tools in use.

Aside from a few inherently interactive tools such as brushing scatterplots and choosing viewpoints, we emphasize images recorded frame-at-a-time onto videotape. Sound works effectively for presenting scalar information in sync with field displays, for adding tick marks on the time axis, and for more subtle stretched data displays.

1. tensor/scatter tools. We have been involved in the construction of algorithms and software for simulating complex physical systems for many years. As simulations in two and three (or more) spatial dimensions and time become more commonplace, manipulating and understanding the results have become important aspects of the overall scientific-computing problem.

It is necessary to switch from a subroutine library mentality to making use of self-descriptive file formats when large simulation tools (run on specialized computer hardware) are involved. We introduced such file formats for tensor-product and scattered data elsewhere[5]; unlike specialized binary formats, our approach uses simple ASCII files that can be processed using an AWK-like paradigm[1]. In cases where speed of input/output becomes a serious issue, we accelerate the ASCII file using [10]. Such an approach has allowed us to build a family of simple tools that can be applied in numerous combinations using standard UNIX^{TM} pipes.

We provide tools to: compute the domain and range of the data; scale data; generate variation-diminishing and least-squares splines in multiple dimensions; provide ubiquitous function plots; generate orthographic and perspective color-level plots; compute sounds suitable for inclusion in scientific videos to demark time and some other scalar parameters. In later sections of this paper, we describe in more detail some of the other tools that we have found valuable. We have found that building small tools has made it possible to move from one hardware platform to another with minimal effort.

2. Slice and dice. One of the more innovative aspects of our function displays is the *dashed surface*. The scattered data smoothing program *loess*[3, 9, 4] produces not only a data model but also an estimate of standard error at each point. The obvious generalization of error bars would be transparent offset surfaces, but this gets rather cluttered. Instead, dice the surface into patches that get trimmed as the error increases. This is analogous to a univariate function plot in which the function curve is dashed in areas of uncertainty. The technique is straightforward to implement, though one needs to be sure to scale the trimming so that patch *area*, not diameter, is proportional to standard error.

So far, the tools described have mostly been applicable to functions of two space variables. Animation is used for a third variable, such as time in a simulation of transient phenomena, smoothing parameter in data analysis, or continuation parameter in a stability study.

[*] Typeset on July 31, 1991.

[†] AT&T Bell Laboratories, Murray Hill NJ 07974 USA. wmc or ehg@research.att.com.

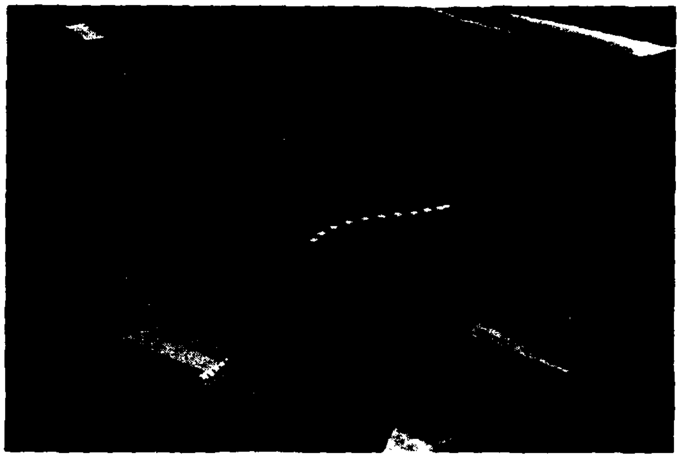


FIG 1. *Continuation example, illustrating icons on transparent isosurface, with notched cube and color contours as reference frame.*

For functions of three space variables, our goal is to emulate the outstanding pictures one finds in a good anatomy text. First we want to cut away parts of the three-dimensional domain, not just with a single cutting plane but making a rectangular notch, sometimes stopping the cut along a curved internal surface. Sometimes this means a blood vessel or the like is left intact in the excavated region. Second, color and surface texture increase contrast on the internal surfaces thus revealed. Finally, distinctive objects (*icons*) are inserted to point out important features.

So far we have taken only the first steps toward this goal. In one VLSI simulation we were recently involved with, the operating envelope of a device was to be computed by holding one voltage fixed, varying a second by an ordinary continuation procedure, and searching for a turning point. Then we varied the second voltage and traced out turning points in the first voltage using predictor-corrector continuation applied to an augmented nonlinear system. As Rheinboldt has analyzed in detail[15], this amounts to tracing a curve on a certain manifold and we have gotten some insight into the behavior of the continuation procedure by displaying this manifold.

A segment in the videotape shows a continuation example of this kind; a still from this is (inadequately) shown in Figure 1. We start with a scalar function of three variables. Although we plan to borrow a BSP-tree program[13], *sculpt*, to specify arbitrary slicing, for now we make do with a simple notched cube, and display by color bands the values of the function on the visible faces. This gives an overall impression of the function to the trained eye, and provides a frame of reference for the isosurface, which is the equilibrium manifold for this continuation problem, and the key item of interest.

Successive turning points found in the numerical continuation are marked by small cubes. The surface is slightly transparent so that the continuation path can be at least roughly seen from any angle. (This presents a technical difficulty, because the z-buffer algorithms used for hidden surface elimination do not properly deal with transparent surfaces in general. In this particular use, we can arrange to compute the pieces of the isosurface in back to front order and sidestep the difficulty.) Both the isosurface and the cubes are given a non-spectral color to contrast with the background. Even so, a proper vantage point is essential for a clear view of the action. So in the animation we pick a camera position using the surface normal at the continuation point just computed. thus encoding some curvature information in the camera sweep.

At present we compute the isosurface by the customary linear interpolation. By adding surface normals at polygon vertices. obtained from a trivariate quadratic variation diminishing tensor spline approximation. visible artifacts are much reduced. This gives a smooth appearance without the need for a very fine grid. However. it does not get around the fundamental limitations of linear interpolation near singular points. We have successfully used Rockwood's program[16] on an experimental basis. but need to adapt it to generate patches instead of points before we can use it routinely.

3. Sound and color. Yet more information can be conveyed in this animation of the continuation process by using the sound track. In our example. we used a snare drum to indicate each Newton step. and a bass drum to indicate a converged turning point. Upon playing the videotape. we noticed for the first time by hearing an abrupt change in the sound track that the convergence got into trouble just as the continuation switched variable 1 to variable 2. Examination of the detailed numerical output confirmed this. giving us a valuable pointer of where to direct further numerical efforts.

In other animations of simulation results. by listening to the sound track we have discovered restarts in the differential equation solver that had earlier gone unnoticed. This was caused by a phenomenon related to stretching the x axis in line drawing graphics. Sometimes small features that can't be seen when the entire dataset is displayed at once become apparent when the scale is stretched and only a small segment of the dataset is displayed at a time.

Even when sound does not lead to a discovery in the data as dramatic as this. it can serve a useful role as a tick mark for the time axis. This could alternatively be achieved by printing the time or. better. drawing a clock face. in the corner of the screen. Sound has the advantage that the viewer's eye is not drawn away from the main action. Animations go by rather quickly until one has viewed them many times. so this advantage of sound (like the head-up display in aircraft) can be valuable.

For further discussion of synchronizing sound and video. see [6]. That paper also describes a numerical procedure for automatically choosing colors for contour bands. Nonlinear least squares applied to psychophysical metrics can eliminate a manual task that is otherwise tedious and arbitrary. For preliminary efforts to correct loudness similarly. see [7].

With all the fascination of elegant renderings and snazzy sound. it is easy to lose sight of the fact that simple line drawings are still the most important graphical tool we have. Probably 80% of our displays come from a tiny library of function and data plotting routines[11]. These generate files in an equally tiny language that can be trivially converted to PostScript. pic. or whatever other graphics is locally available. For dependency graphs and other such figures. some useful tools are MetaPost[12] and dag[8]. PC-based drawing programs are also attractive. but do not lend themselves to data-assisted creation and updating.

One line drawing that deserves wider use is *brushing scatterplots*[2]. This technique for displaying sets of multivariate data points presents a matrix of all pairwise scatterplots and allows the mouse to paint points in one plot while simultaneously highlighting the corresponding points in other views. By this means. a number of patterns and outliers can be quickly identified.

4. Hardware-related choices. Not so long ago, graphics applications tended to be tied to specific hardware. The reason was not lack of standards in calling sequences and file formats, but rather in fundamental differences of hardware architecture and the need to squeeze the last ounce of performance out of a system in order to get acceptable speed. On one system, color-band contour plots would be produced by operating on the color lookup tables: on another system, using a texture map proved more effective. Fortunately, the situation has changed and we are now blessed with comparable functionality in all the high-end graphics devices. Even the lower-cost devices have plenty of performance for common visualization needs. However, hardware considerations continue to force important architectural decisions in graphics software.

High resolution color workstations (1280 by 1024 pixels, 8 bits deep) seem to be the current norm, but we prefer to work with smaller image buffers (640 by 480 pixels, 24 bits deep). This is a strategic decision: if you develop applications that depend on high resolution, until HDTV arrives you'll find it very difficult to produce a legible videotape. How often we hear the lament at conferences, "Well, you can't see it very well here, but back in my lab..."! Working at the lower resolution forces one to come to grips early on with proper antialiasing, integrating it well into the environment instead of leaving it as a last-minute hassle when making photos under deadline pressure.

The NTSC discipline also discourages the application writer from filling the screen with menu and dial widgets. We prefer to relegate these to an adjacent monochrome X display or, even better, to prune the user interface ruthlessly so no widgets are necessary. For example, we adopted a helicopter model for all our tools, so that the mouse is used as a virtual control stick for flying around the object on display. Immediate visual feedback works more smoothly than twirling a dials attached to each axis. As another example, Pat Hanrahan's *medit* program lets one move lights around by grabbing a reflection highlight with the mouse and moving it on the surface of a displayed object. To our eyes, a control widget on-screen is like chartjunk[17] in a line drawing; it may serve some valid purpose, but the overall effect is to distract from the data.

For portability, we have restricted ourselves to a three-button mouse as the only input device. There is some concern also that adding degrees of freedom overwhelms the average operator. An interesting lesson in the development of V/STOL aircraft was that pilots learned to take off more stably by being trained to only change a few throttle and attitude variables at a time.

The issue of 8 bits versus 24 bits is less decisive: there are effective dithering and color compression algorithms available to squeeze images into 8 bits. But these are global algorithms and lead to visible artifacts unless properly tuned, so we prefer the simplicity of 24 bits, made affordable by the astonishing drop in memory prices. The image needs to be rendered anyway at 24 bits (or more, since alpha and depth values are also commonly needed). The 640 by 480 resolution, in contrast, does yield a noticeable improvement in rendering times.

Another advantage of standardizing on NTSC is that displays are inexpensive enough to put on everyone's desk, and signals can be readily distributed by existing video switch technology. Graphics tends to be a tool that is just used a few times a day; when one has to run down the hall to a graphics lab, it tends to be used less than it should be. One solution is to put personal graphics workstations in each office, but that is expensive. Making the displays cheap allows one to purchase fewer, more

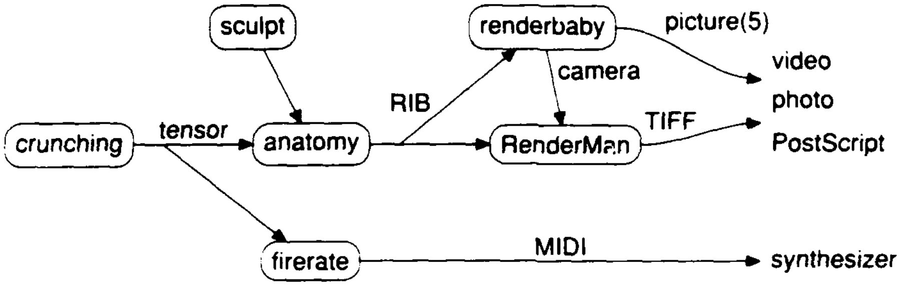


FIG. 2. Software pipeline, with the "main path" consisting of batch-mode computation culminating in synchronized audio-videotape. The "anatomy" filter is composed of several independent programs reading spline coefficients in "tensor" format and producing geometric objects in "RIB" format

powerful graphics engines to be shared among more people, some of them just casual users who can't justify the cost of a dedicated workstation.

Even with an investment in high-end graphics workstations, the rendering time for many of our displays may be several seconds per frame. Rather than compromise on the graphics or spend great effort optimizing the rendering, we content ourselves with making a video in batch mode, recording animation frame-at-a-time. The numerical simulation that feeds the graphics is typically run in batch mode anyway, so this does not feel very restrictive.

Our top level pipeline is shown in Figure 2. RIB (RenderMan Interface Bytestream) [14] is a convenient language for describing geometry. It has enough generality that all visualization techniques we can currently contemplate using are covered; it makes gluing together the output of several independent programs easy; it can be rendered by high-quality, commercial software or (in restricted subsets) by simple home-brew drivers adapted to local hardware. TIFF and picture(5) are two image file formats. MIDI is a standard music file format.

The economics of publishing at the present time preclude the use of color for specialized scholarly journals. Since animation is at least as valuable to us as color, we choose to skip from PostScript^{TM} drawings straight to video, bypassing the difficulties of printing accurate color.

We thank Allan Wilks for contouring code and for comments on a draft of this paper. PostScript is a registered trademark of Adobe Systems Inc. RenderMan is a registered trademark of Pixar. UNIX is a registered trademark of UNIX System Laboratories, Inc.

REFERENCES

[1] A. V. AHO, B. W. KERNIGHAN, AND P. J. WEINBERGER, *The AWK Programming Language*, Addison-Wesley, New York, 1988.

[2] R. A. BECKER AND W. S. CLEVELAND, *Brushing scatterplots*, Technometrics, 29 (1987), pp. 127–142.

[3] W. S. CLEVELAND, S. J. DEVLIN, AND E. GROSSE, *Regression by local fitting: Methods, properties, and computational algorithms*, J. Econometrics, 37 (1988), pp. 87–114.

[4] W. S. CLEVELAND AND E. GROSSE, *Computational methods for local regression*, Statistics and Computing, 1:1 (1991).

[5] W. M. Coughran, Jr. and E. Grosse, *A philosophy for scientific computing tools*, SIGNUM Newsletter, 24:2/3 (1989), pp. 2–9.

[6] ——, *Techniques for scientific animation*, SPIE Proceedings, 1259 (1990), pp. 72–79.

[7] ——, *Seeing and hearing dynamic loess surfaces*, in Interface'91 Proceedings, Springer-Verlag, 1991.

[8] E. R. Gansner, S. C. North, and K. P. Vo, *DAG — a program for drawing directed graphs*, in Unix Research System Papers, Tenth Edition, Volume II, Saunders College Publishing, 1990, pp. 147–162.

[9] E. Grosse, *LOESS: Multivariate smoothing by moving least squares*, in Approximation Theory VI, C. K. Chui, L. L. Schumaker, and J. D. Ward, eds., Academic Press, New York, 1989, pp. 299–302.

[10] ——, *How shall we connect our software tools*, in Visualization'91 Proceedings, IEEE Computer Society Press, 1991.

[11] E. H. Grosse and W. M. Coughran, Jr., *The pine programming language*, Numerical Analysis Manuscript 83-4, ATT Bell Laboratories, 1991.

[12] J. D. Hobby, *A MetaFont-like system with PostScript output*, Tugboat, the TEX User's Group Newsletter, 10 (1989), pp. 505–512.

[13] B. Naylor, *SCULPT an interative solid modeling tool*, in Graphics Interface '90, Canadian Information Processing Society, 1990, pp. 138–148.

[14] Pixar, *The RenderMan Interface, Version 3.1*, 3240 Kerner Blvd. San Rafael CA 94901, 1989.

[15] W. C. Rheinboldt, *Computation of critical boundaries on equilibrium manifolds*, SIAM J. Numerical Analysis, 19 (1982), pp. 653–669.

[16] A. Rockwood, *Accurate display of tensor product isosurfaces*, in Visualization '90 Proceedings, A. Kaufman, ed., IEEE Computer Society Press, 1990, pp. 353–360.

[17] E. Tufte, *Visual display of quantitative information*, Cheshire, Conn., Graphics Press, 1983.

Distributed visual programming environment: an attempt to integrate third generation languages with advanced user environments.

Frank M. Rijnders*, Hans J.W. Spoelder*, Frans C.A. Groen*#,
*Free University
Faculty of Physics and Astronomy
Department of Physics Applied Computer Science
De Boelelaan 1081, 1081 HV Amsterdam,
The Netherlands
#University of Amsterdam,
Faculty of Mathematics and Computer Science
Department of Computer Systems

Abstract.

This paper describes a visual environment designed to easily develop programs based on already existing software. Typically third generation scientific libraries (e.g. IMSL, NAG) having procedural interfaces should be incorporated. The design supports distributed execution of programs in a dataflow model, offering the possibility to allot software procedures to suitable architectures. Attention in this paper is focused on design problems originating from the great variety of data-structures that have to be supported.

1. Introduction.

With the advent of high resolution graphics workstations a significant impulse has been given to the development of general tools for advanced user interaction and program- and data- visualization. On the one hand, important achievements in the field of operating systems and associated toolboxes (Mac, Sun) need no further emphasis. On the other hand, compared to this, developments of tools for program and data visualization are complicated due to the broadness of the field of possible applications. (For a review we refer to (Shu89) and (Vis86)). In our view, a visual environment designed for general scientific problem solving should excel in two major properties besides visual representation, namely easy incorporation of already existing software and the possibility for distributed execution. The first quality is important since the different scientific communities heavily use existing software in the form of well maintained and highly optimized libraries like IMSL, NAG, ESS1 and own problem specific libraries. This quality promotes software reusability. The latter quality is necessitated by the trend of current computer facilities showing a high degree of architectural differentiation (Ros89) for activities like 'number crunching', data acquisition and visualization. Other basic concepts to be incorporated are: visual representation of programs and advanced user-interaction for program development and execution. In this paper we discuss a distributed visual programming environment under development at our laboratory (Acm91), and focus on some problems encountered and their possible solutions.

In section 2 we discuss the strain between a data oriented and a procedure oriented approach. Section 3 describes the environment design and some aspects of the implementation. Section 4 focuses on the approach taken for supporting diverse data-structures within the environment.

2. Data object versus procedure object oriented approach.

Roughly speaking the transition between traditional third generation programming languages and advanced systems is marked by the introduction of the concept of data objects, i.e. a user-defined collection of data and a set of associated functions by means of which the data can be manipulated. The concept of data object is especially apt for visualization of data (see for instance Ups89).

A case study, Signor (Cort89), in the field of systems and signal theory, was carried out within our laboratory. In this case the data-set comprised one dimensional arrays, representing a signal in a finite time interval, whereas the actions included operations like generation of data, transformation filtering, deconvolution etc. The visualization of data consists merely of graphs of value versus time for this restricted data set, while program visualization is done by iconic representation of the actions. Thus, in Signor a restricted data set combined with a restricted set of actions defines a powerful visual signal processing language.

One of our major objectives, to support existing software coded in some third generation language, rules out this possibility of defining an entirely new visual environment having a well defined set of data objects, since all existing software should then be adapted to meet the specific requirements of this new environment. Instead, it calls for a procedure oriented approach in which the environment is adopted to the requirements imposed by the procedures, i.e. the basic object in our environment is related to a procedure which

Acknowledgement: Part of this research was funded by a grant of IBM, The Netherlands.

transforms the data (Acm91).

The combination of the concept of using existing software procedures as atomic objects and the concept of visualizing programs defined in terms of these objects implies the use of a dataflow model(Dav82) for execution of the program. This is argumented by the following reasons: firstly, solving scientific problems in terms of already existing modules puts the flow of data more central than the flow of control in a program. The fine grain flow of control will be found inside the procedure objects, screened from public access. Here optimization with respect to the specific architecture like vectorization and/or parallellization can be carried out. The course grain flow of control is determined by the data dependencies among the parameters of the different procedures, so it is the dataflow that governs the program execution state. This is especially important for the control of distributed processes.

The second reason to use a dataflow model is that flow programs are closely related to visual programs: they are easily translated into graphs, the nodes representing objects (procedures) and the arcs representing data dependencies between procedures. This graph defines a visual representation of the program. Another advantage is that flow programs easily combine into larger programs, offering the possibility of layered programming. Viewed from the implementation level, a distributed execution model based on a dataflow is (relatively) straightforward.

It should be stressed here, that the abstraction level of the atomic objects (the nodes in the graph, representing 3gl library procedures) in our flow model will in general be considerably higher than the operators in a regular flow language, although low level procedural statements are not excluded (and are probably often indispensable). Clearly we intend to reduce the number of low level statements needed in relation to data conversion and conditional execution in programs by offering advanced facilities in our environment.

During the last ten years, the object oriented model has found wide acceptance (Cox86). Object based models were applied for constructing distributed systems [Bir85, Bla87]. Object orientation provides natural ways for composing a large system from a number of autonomous components which interact only via some well defined interface. Our approach to represent the 3gl procedure as the atomic objects in a visual environment clearly saves a number of the advantages of the object oriented model, in particular at the environment level but not so much at the flow programming level. Like in object oriented programming, the atomic- or procedure object in a dataflow environment behaves as an autonomous entity that has a state. It responds to messages by its data dependencies. With the proper rules for creating compound objects, neat classification and inheritance mechanics are imposed. Regarding encapsulation, compound objects may very well hide parameters that are not of public interest. However, there is a leeway between objects in our environment and objects in the object oriented model, caused by the fact that in our approach the procedures action is central, while in the object oriented model the data is central. One could paraphrase this as a strain between procedure-objects and data-objects, were in the first case procedural action is governed by dataflow, whereas in the second case data is governed by procedural interaction. No doubt this discrimination is artificial and depends on a relative point of view, but it serves as a useful concept here.

Procedure oriented approach

Procedure objects are represented in the environment by icons having definable images clarifying their action. This action is commonly defined by a third generation language procedure.

A major problem of the procedure oriented approach is that the data exchanged between procedures can have virtually any structure as conceivable in the third generation languages supported. Although most commonly used datastructures will be uncomplicated, sophisticated support of general structures is a necessity if the environment has to meet the requirements of the scientific user, because we aim at viable mixing rather heterogeneous sets of procedures. This support includes edit and type-cast facilities for arbitrary data-structures. Among the problems that had to be solved in this context is the architectural dependency of data representation: not only does the representation of basic types (i.e. C- language floats or doubles) differ, but also the way compound types are composed differ. The latter is caused by alignment differences but can also be language (or even compiler) dependent.

The procedure-object approach offers the possibility to (re)group a number of procedures to a new composite object. This may prevent the visual representation of growing visual programs (containing many objects) from cluttering the screen. Composite-objects have their own resulting parameters and iconic representations. They represent sub-programs, thus offering a way to structure the visual program in layers.

3. Environment design

The environment must provide all that is necessary to edit and execute programs. There are five more or less isolated aspects: objects have to be selected (by a query tool), objects have to be displayed and manipulated for program definition(by a program visulalizing tool), object's parameters have to be edited (by a so called varedit tool) and objects have to be executed (by a runtime execution model, controlled from the visualizing tool). Lastly, some tool is needed to easily import new procedures in the environment.

Because the wealth of existing libraries results in numerous objects, a 'selection by query' tool greatly simplifies this selection by presenting the objects in an organised form.

A program in this visual environment is visualized by icons and connections (arrows) resulting in a two dimensional directed graph. We will refer to the user-interface concerned with this visualization as the program visualizer. The nodes in the graph are procedures represented by object icons. After selection, these icons can be placed on a draw area in the program visualizer (see figure 1). Connections (edges) in the graph are created by drawing lines from procedures producing output data into procedures accepting input data.

When executing a visual program, its graph is mapped onto a dataflow model of the program, thus the connections serve to represent the dataflow. It should be pointed out here however that the program description, residing in a database, could also be passed to a code generator as an alternative. This code generator could produce ready to hand, compiled, program modules that execute faster compared to the flow model alternative. It will be clear though, that the flow model is far more flexible when steering execution or adjusting the program during execution.

Another essential user-interaction is the process of editing procedure parameters. This edit interaction must have the capability of visualizing generic data-structures, in that providing for the needs of easy browsing and updating of (possibly) complicated data-structures. This user-interaction requires (in common with, among other things, the flow connection action), that a distinction must be made in the visualized program between the different parameters of an object. Therefore, the objects icon area is subdivided into one or more sub-areas. These sub-areas serve as sensitive spots during these user-interactions and, being definable small icons themselves, they may depict the data-structures involved. Consequently, in order to edit a parameter, the appropriate action-event is selected from a menu with the aid of a locator. Subsequently, this event is dropped on the sub-area representing that parameter. This interaction will start a parameter edit session in a separate window. In this window, the parameter's data-structure is utilized to present the data involved. The user can access different parts of the data-structure either by specifying names or by simply selecting names with the locator.

Requirements with respect to storage and management of data is an issue in the implementation of this environment. In this context, data refers to all conceivable forms of data involved like: icon images, values and definitions of parameters, procedure names, but also the program graph descriptions. At this point we take advantage of the capabil-

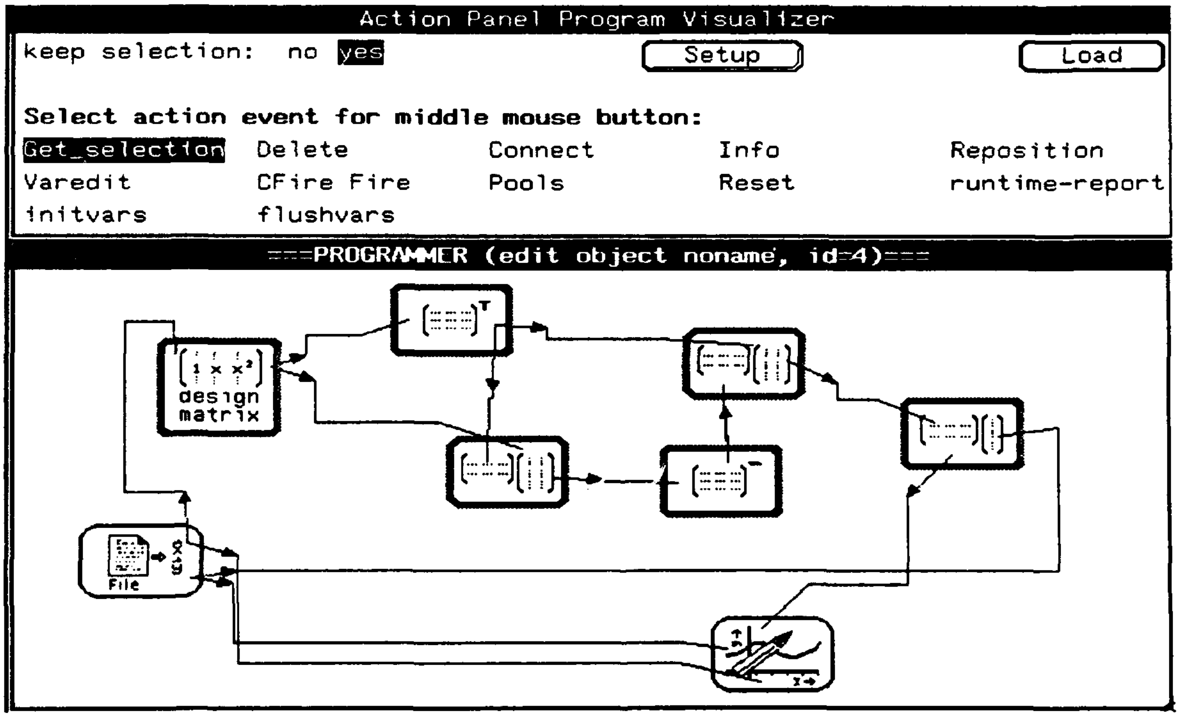


Figure 1

Example program representing a least squares fit that fits a polynomial and plots the source data and the fitted data. The formula used is $\theta = (X^T X)^{-} X^T y$

Dropping a Varedit action event on the design matrix object will reveal its only remaining non-private variable: the desired polynomial degree, which can subsequently be edited.

ities offered by relational database management systems (rdbms). Other implementation key issues are execution of programs and the management of data-structures during creation and execution of programs.

Execution of programs

Any object (whether atomic or composite), or any collection of objects and connections, defines an executable program in the environment. This means that any object found with the query tool can also be executed.

This stand-alone execution of objects offers the user the possibility of getting familiar with available software. This is only meaningful when the environment has default values available for the input parameters involved: many procedures found in software libraries like IMSL need numerous parameters that are not at first of interest. In the current implementation, parameters can be supplied with default and extreme values.

The use of a dataflow model for executing a visual program has yet another advantage: the fact that parameters are participating not only in the data- but also in the control flow of the program, offers handles for visual feedback from the execution state of objects. The combination of an environment, visualizing 3gl procedures as objects, and executing of those objects in a dataflow model, presents the executing program in an object oriented way: The user can query (running) objects on their execution state, input values and resulting output values, from the program visualizer interface. This is simply done by selecting the relevant event (e.g. a request status event) and passing this event to the object in question by a subsequent mouse click on the object's icon image.

For implementation of the dataflow model we used a client/(multiple) server combination based on the remote procedure call (RPC) library (this software library offers the mechanics for remote execution of procedures). In this case the client is the program visualizer, giving the user complete control over the dataflow. The purpose of the servers is to execute the code involved. Each server functions as an encapsulation of a set of procedures and is generated and forked on the site in the network where those procedures were intended to be executed. The encapsulation handles the administrative tasks involved in the dataflow model, i.e. how and when to fire (execute) an object. Moreover, it serves as a framework for exchange of data between objects. This exchange is based on a runtime description of the memory layout of the procedure's parameters, which is discussed in the next section. As intended, the client/(multiple) server design allows for network wide, highly parallel, execution of the program. Clearly it is also possible to incorporate in this way all kinds of different dedicated hardware, on the condition that remote procedure call (RPC) software is supported.

To execute a flow program three steps are needed. Firstly, encapsulation servers are generated based on the configuration of procedures used and sites in the network participating. Secondly, these encapsulation servers are executed (forked). Thirdly, the encapsulation servers are loaded with all needed information from the database, like initial values of procedure parameters and destinations of output variables. In order to direct the first and second actions, every site participating in the network runs a daemon called creation server. Its purpose is creating, controlling and destroying the encapsulation servers. Besides, it reports back on valuable information from its host, such as the current workload. These three setup actions are automated and are activated by the user by means of a setup-menu from the program visualizer.

At runtime, all communication imposed by the dataflow directly takes place among the encapsulation servers for efficiency reasons. The program visualizer interface though, is at any time in control of the state of the encapsulation servers. That is, it can hold, release, fire or reset objects in the executing program.

4. Mixed language data-structure description.

The procedure oriented approach calls for generic definition of datatypes involved due to the variety of procedures the environment supports, like for instance fortran coded math libs or c-language coded user-defined procedures. Such in contrast to specialized environments (like Signor) where the set of datatypes is limited and can therefore be handled on an individual, hard coded, basis.

The data-structure description used is designed to support, in a transparent way, different architectures and to allow, in a flexible way, a typecast from one datatype to another. Whereas the first characteristic is a mere necessity the second characteristic augments user-friendliness to adjusting data-structures used by various application procedures, without having to write explicit code.

By defining a data-structure description method, a standard is set for formatted storage of data in files, strings and databases. This standard is heavily utilized in low level software layers in the environment design, but it is also applied at the user level by predefined objects, that handle (store, pack, retrieve etc.) structured data. For example, one can think of an object recognizable by its icon depicting a database: on reception of data, the object also receives the data-structure description and utilizes this description to generate a record layout that will fit the data-structure.

The data-structure description is a superset of datatypes defined in those programming languages which are presently supported. We have implemented a recursive description, defined by a list of tokens, which decomposes the user-defined data-structures into basic types like chars, integers and floats (see table 1). This tokenlist is designed for fast decomposing data-structures residing in memory, thus the list serves as a description fit for use at runtime. The distinct memory layouts generated by different types and languages are reflected in the set of available tokens. To screen users from this (relatively) low level structure definition, a simple (bi-directional) parser is invoked producing the recursive type-definition when entering or editing type-definitions. It has its own pascal styled syntax, but for instance, a preceding c language keyword 'typedef' will allow c type-definitions to be used.

To allow an easy user-interaction (e.g. when using the varedit tool), besides the data-structure description logical names are needed. Therefore, detached from the tokenlist, the parser generates a packed namelist for these occasions.

As a simple example, we will consider a C-language defined matrix versus a fortran defined matrix. From a users pcint of view, there should be no difference in usage, so the environment presents both matrices in the form of a column of rows when displaying or type-casting. This implies that type-casting one into another yields the expected conversion (i.e. transposing the matrix). This principle of a meaningful (default) type-conversion, extends to other stuuctured datatype descriptions including for instance C-defined matrices defined by a column of pointers to arrays.

At the implementation level though, memory layouts differ. In the C-language, a matrix is stored row-wise in memory, whereas in fortran a matrix is stored column-wise in memory.To handle these situations in a correct manner, the describing tokenlist knows different tokens for row-wise versus column-wise memory decomposition. The tokenlist

_array n _array m subtype

is in accordance with the memory layout of row-wise stored matrices because the decomposition defined by this list assigns the _array tokens more to the right side higher priority for changing their index values. In contrast, in the tokenlist

_farray m _farray n subtype

higher priority is assigned to the _farray tokens that are more on the left in the list, in this way mapping on column-wise stored matrices. When entering definitions, this difference is implied in the usages of square brackets for row-wise versus round brackets for column-wise storage.

The parser generating the tokenlist allows for both the usage of premeditated- and the usage of default names. For instance, the afore mentioned tokenlist would be generated by parsing the definition

matrixname[n] of rowname[m] of
elementname of subtype;

as well as by the definition

matrixname[n,m] of subtype;

The first definition specifies names at every level in the recursive specification: an array-element name, a name used when addressing rows, and a name used for addressing the whole matrix. This guarantees that the object-designer fixes the names that the environment will present during user-interaction. The second straightforward definition attaches default names to sublevels of the data structure.

Table 1:

compound datatypes (#: number of elements):
type: basic type | compound type.
Compound type:

array	*#*	*subtype*
farray	*#*	*subtype*
struct	*subtype {, subtype} endstruct*	
complex	*subtype*	
rptr	*subtype (relocatable pointer)*	
ptr	*subtype (non relocatable ptr)*	

examples of basic types are:

int4	
float4	
float8	
string	
chararray	*#*
bytes	*#*

Software layer for runtime support of the data-descriptions

The software layer implemented for supporting the datatype descriptions is capable of runtime composing and decomposing memory areas according to the tokenized datatype description. As pointed out, this decomposition is a key issue at the implementation level. Yet another example of this is the usage of the (de)composition in combination with basic calls to the XDR library (this library converts data-structures to and from an external representation), in order to exchange structured data among different architectures. To solve the problems originating in the architectural and language dependent alignment differences, the tokens in the tokenlist are packed with additional information. This information is generated on creation or receipt of a tokenlist on a specific architecture; the information being dependent on the specific architecture as well as on the configuration of the tokenlist involved. The information in question mainly consists of (cumulative) size information of subtypes in the context of the spanning type.

How the decomposition is used is briefly illustrated by the flow diagram in figure 2. The decomposition is done by a procedure altering a private stack. The stack contains all relevant data like address pointers in the memory area being decomposed, the current type being accessed etc. After every iteration of this procedure, the next basic element (if any) is located in the memory area and in the tokenlist, and is handled by examining the contents of the stack. If flags on the stack indicate that initialization, termination, or separation events occurred during the iteration, proper callback functions set previously by (and thus depending on the nature of) the caller, are invoked. For exampie, these callback functions typically print colons, opening and closing brackets in print functions that are designed for printing structured data (after

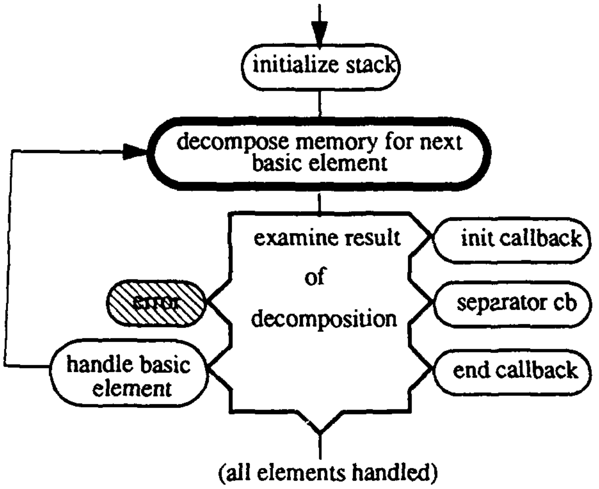


Figure 2
Decomposition of memory areas is done by iterating a decomposition procedure and examining its results. The datatype structure, a memory address referencing the memory area and the proper callback functions are set in the init phase. Special ways of decomposing structured data, needed when typecasting, can also be specified in the init phase

the manner of initialization assignments for datatypes in the c-language). But, depending on the type of the caller, more sophisticated tasks can be done, like memory (re)allocation during typecasts, or when reading (e.g. from files) into pointer typed variables.

Typecasting is done by the environment by parallel decomposing the source and destination data-structures. The basic source elements will be converted to the basic destination elements, the afore mentioned callbacks performing typespecific actions, e.g. when casting to an array pointed to by a nill pointer, the initalization callback invoked will perform the implied memory allocation.

Extending the set of types supported

In developing this advanced environment, we aimed at creating a fast prototype environment besides supporting existing 3gl procedures. In this context, the environment has two types of end users: users who will exclusivly use already existing procedure objects and users who will also extend, modify and maintain a collection of procedure objects. The latter requests that the set of datatypes supported is extendable. By extending the set of basic data types, types can be introduced having special properties regarding typecasting and other basic interaction.

As an example of the power of special properties attached to special types, consider the chararray type from table 1. This datatype maps on the same memory layout as the (structured) datatype: array of chars. However, casting an integer type to the chararray type will result in a string representation of the integer, in contrast to casting an int type to an array of chars. The latter would result in a normal cast of an integer to the first element in the array, being of character type. The complex type in table 1 serves as an example of a special compound type. Most common, its subtype would be doubles or floats. Type specific actions in this case typically affect ascii printing and reading style (e.g. use of the form a+bj). Moreover, type-casting between complex variables having different subtypes is transparent now.

Towards a visual flow language

It will be obvious that the proposed environment promotes the making up of organized collections of objects. These collections can be viewed as object libraries. While on the one hand these libraries will naturally emanate from imported 3gl libraries, being topic specific, on the other hand there will be special purpose libraries. In the latter category are libraries with objects performing actions that transcend regular procedural action, for instance in that they affect the dataflow during execution. This will typically be reached by doing calls to the flow system, comparable to system calls done to an operating system.

As an example, a so called convert object alters the flow in a flow program: It collects input data and converts this data to (commonly different structured) output data. In this way a conversion is achieved that (de)serializes a data stream to (from) a stream of other structures. One can think of an object producing numbers, that are collected in an array by the convert object. The array can subsequently be passed to some display object that plots a graph.

Another example is a special conditional (flow) 'if' object (figure 3) having a chameleon typed input attribute, meaning it copies a received data structure description as its own structure definition. This object has, besides its input attribute, a function attribute and two output attributes. The function attribute can be connected to an object that will receives the afore mentioned data structure and returns a decision value for the 'if' statement. The data and its structure are propagated to the output corresponding to the result of the 'if'. For example, the object interconnected with the function attribute of the 'if' object could evaluate a determinant; the 'if' object would decide on the resulting value whether the dataflow should be passed to a matrix inversion object or not. The point in describing these examples is that a well-consid-

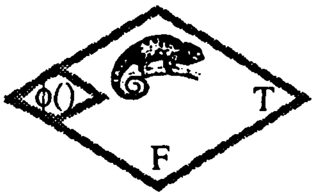


figure 3.
Representation of an 'if' object propagating data to one of its outputs, depending on the result of an external function.

ered collection of special purpose objects, can very well serve to constitute a visual (flow) language, but again we have a higher object abstraction level in mind than a regular 3gl statement level.

5. Conclusion.

The purpose of our visual programming environment is to facilitate the programming effort which a scientist has to make, when composing a program consisting to a large extent of already existing (often problem oriented) software. The environment which we have developed can elegantly be used for this activity. Its ability for distributed processing allows the user to utilize nodes in the network without specific knowledge about their architecture, operating system or details about remote procedure call mechanisms. Compared to the traditional programming activity the user now is mainly concerned with selecting the appropriate software and defining the intended data exchange, without having to explicitly code common actions like variable editing and typecasts. The possibilities of the cast operation allow for intermixing procedures, each introducing their own set of datatypes.

Taking together these advantages, the user can concentrate on his specific problem at a more abstract level, heedless of low level implementation details. This simplifies prototyping and program development. This visual environment adds considerably to the possibilities of line oriented debuggers by offering stepwise execution control and accessory data retrieval at the abstraction level of the problem. The user can selectively and interactively stipulate which and when data is to be visualized, either by simply requesting a structured rendering of the data or by linking the dataflow into some suitable procedure object capable of visualizing the (possibly casted) data.

Currently we are testing the prototype on a number of typical applications to measure the reduction in programming effort achieved by using this environment and the efficiency of the client/server dataflow model and its possibilities in the field of course grain parallelism.

Among the aspects we have to gain more experience with, is how the contrast between our procedure oriented approach and a data oriented approach works out in practice. Our experience with solving problems using topic specific libraries is promising, but in these cases the libraries already imply the use of high level abstract data objects (e.g. matrices). This is clearly different from using existing problem specific code not originally designed in a more or less (data)object oriented way. Another aspect we need to gain experience with is the impact of the relational database management system used in the implementation of the environment, especially with regard to the time needed for loading large programs in the visualizer and in the execution model.

Also, there are a number of problem fields that have yet to be investigated. One of these problems is the conflict between usage of fortran common blocks and distributed execution: clearly, restrictions imposed by the use of 3g language procedures are not yet fully solved here (that is, they are not yet completely transparent to the environment user). Similar problems are encountered with respect to file io handling in different 3g languages.

References

Mac Inside Macintosh, Apple Computer Inc., 20525 Mariani Avenue Cupertino, CA 95014.

Sun SunView system Programmers Guide, Sun Microsystems Inc., Mountain View CA, 1990

Shu89 N. C. Shu, "Visual Programming", Van Nostrand Reinhold Company Inc., Princeton 1989

Vis86 S. K. Chang, T. Ichikawa, P. Ligomenides (eds), "Visual languages", Plenum Press, New York, 1986

Acm91 F.M. Rijnders, H.J.W. Spoelder, E.P.M. Corten, A.H. Ullings, F.C.A. Groen, 1991 "Versatile visual programming environment for scientific applications", Proceedings of the ACM Symposium on Personal and Small Computers, Toronto 1991.

Cort89 E.P.M. Corten, H J.W. Spoelder, F.H. Ullings, F. C.A. Groen, "Signor: a tool for the visualization of concept of systems and signal theory Proceedings of the IEEE workshop on visual languages, October 4-6, Rome, 1989

Ups89 C. Upson, T. Faulhaber, Jr., D. Kamins, D. Laidlaw, D. Schlegel, J. Vroon, R. Gurwitz, A. van Dam. 1989 "The Application Visualization System: A Computational Environment for Scientific Visualization", IEEE Computer Graphics and Applications, July 1989, p. 30-42

Ros89 Rosenblum, L.J., "Scientific visualization at research laboratories", IEEE Computer, vol 22, pp.68-70, August 1989

Bir85 K.P. Birman, T.A. Joseph, T. Raeuckle and A. El Abbadi, "Implementing Fault-Tolerant Distributed Objects", IEEE Transactions on software engineering, Vol. SE-11, No. 6, June 1985, p. 502-508

Bla87 A. Black, N. Hutchinson, E. Jul, H. Levy and L. Carter, "Distribution and Abstract Types in Emerald", IEEE Transactions on software engineering, Vol. SE-12, Januari 1987, p. 65-76

Cox86 B.J. Cox," Object-Oriented Programming, an Evolutionary Approach", Addison-Wesley, Reading, Mass., 1986

Dav82 A.K. Davis, R.M. Keller, "Dataflow program graphs", IEEE computer, vol 15., no. 2 (feb 1982), pp. 26-41

Visualization and its Use in Scientific Computation

K W Brodlie, M Berzins, P M Dew, A Poon, H Wright
School of Computer Studies
University of Leeds
LEEDS, UK

1 Introduction

The importance of visualization in scientific computation is now widely recognised. Recent advances in large-scale, parallel computing have increased the potential of numerical simulation as a means of experimentation in the applied sciences; visual techniques of data presentation play a key role in the understanding and evaluation of the simulation results.

The birth of the subject is often associated with the the publication of the NSF-commissioned report 'Visualization in Scientific Computing' (McCormick et al, 1987). This report argued the case for increased attention to be paid to visualization, if full value was to be obtained from the investment in the US national supercomputer centres. Several of these centres then initiated major research and development projects in visualization, the results of which are now beginning to appear. A recent survey volume (Nielson et al, 1990) provides a good overview of these activities.

It is therefore tempting to think of 'scientific visualization' as a new discipline. But there is a case for arguing that it is only the term which is new. West (1991) claims that historically many original thinkers in the physical sciences have relied heavily on visual modes of thought, using images rather than words and numbers. For example, James Clerk Maxwell, founder of thermodynamics, built 3D clay models as an aid to his understanding of functions of two independent variables. (Some interesting computer simulations of Clerk Maxwell's work have recently been done (Jolls and Coy, 1990)).

From the early days of computing too, scientists have used computer graphics as a key part of their experimentation. In the UK, much pioneering graphics work, involving the production of animated sequences on film, was carried out at the Rutherford Appleton and Culham Laboratories in support of physicists in the 1960s. The Culham work led to the development of the GHOST system (GHOST, 1982) which has been widely used in the UK scientific research community over two decades.

While one can debate whether scientific visualization is a new subject or merely a new name, there is no doubt its importance has increased significantly in recent years. This can be attributed to a number of key developments:

- The increased computing power offered by modern systems, especially parallel and novel architectures, has extended the range of numerical simulation experiments which scientists can carry out. For example, meteorologists are tackling increasingly large atmospheric models. These simulations in turn generate vast amounts of data - 'firehoses' of data as they have been called; this data has to be evaluated as efficiently as possible. It is simply impossible for the human brain to comprehend more than a tiny fraction of the data in numerical form. However by converting entire fields of variables to a colour image, the brain is able to assimilate global information abnut the simulation;

- The increase in automatic data collection equipment, in particular medical scanners and remote sensing devices, has likewise led to large quantities of data requiring rapid processing;
- The trend away from batch processing to interactive working on a window-based workstation has brought graphics display technology to the scientist's desk as the norm.

While much of the recent activity in visualization has stemmed from the US supercomputer centres, there is an increasing contribution from Europe. A Eurographics working group has held two successful workshops, while in the UK, a recent workshop produced a status report on the subject (Brodlie et al, 1991). The authors of this paper are working in a collaborative research project, called GRASPARC, which is examining the close integration of computation and visualization (Brodlie et al, 1990). The project involves NAG Ltd, University of Leeds and Quintek Ltd.

This paper arises from our work on GRASPARC. Section 2 gives an overview of current work in scientific visualization which has provided input and stimulus to the project. The subject has still to mature - there are different views of its scope and underlying model, and no standards for the functionality to be provided by a system. A small prototype has been built within the GRASPARC project, to help our understanding of the subject and tease out the problems. This prototype is described in section 3, with future plans and conclusions in section 4.

2 Current Work in Scientific Visualization

2.1 Scope and Definition

A number of attempts have been made to define the term 'visualization'. A common flavour to these definitions is the idea that visualization is an aid to understanding: it is part of the experimental process, not simply a means of presenting the final results.

A succinct definition is given by Haber and McNabb (1990):

> Visualization is the use of computer imaging technology as a tool for comprehending data obtained by simulation or physical measurement.

This definition fails however to convey the value of visualization in allowing the scientist to control or steer the simulation, an important aspect certainly in the GRASPARC project. Marshall et al (1990) distinguish three modes in which visualization can be used:

Post-processing where the simulation results are stored and viewed at a later time, the scientist having interactive control over how the data are displayed;

Tracking where the simulation results are fed directly to the graphics module, the scientist again controlling how the results are displayed;

Steering where the simulation results are again fed to the graphics module, but with the scientist having interactive control over both simulation and visualization.

This suggests an extension to the Haber and McNabb definition, along the lines that visualization can be seen as a tool for guiding the computational process.

2.2 Underlying Model

Just as different authors have suggested different definitions of visualization, so there have been several attempts to define a model of the processes involved. A common thread is to distinguish different phases through which a physical problem passes towards its computational solution.

Haber and McNabb (1990) present the underlying model which is the basis of the National Center for Supercomputing Applications (NCSA) RIVERS project. They draw a parallel between the different stages in numerical simulation, and the corresponding stages in scientific experimentation. Thus they separate the simulation into three phases: *modelling, solution*, and *interpretation / evaluation.*

The modelling phase involves a two-step process in which the original, perhaps loosely defined, problem is first posed as a well defined physical model, and is then translated into an idealized mathematical formulation. The solution phase typically involves a discretisation step, where the continuous mathematical model is approximated by some discrete model to which numerical techniques can be applied. The output from this phase is a field solution for the unknown quantities in the model. Finally the interpretation and evaluation phase involves the analysis of the computed results, leading perhaps to some modification in the physical model, its mathematical idealization or the numerical solution technique. It is in this final phase that to date visualization has been most successfully employed, taking the field solution and displaying it in different ways.

Haber and McNabb see visualization itself as a three stage process. These stages transform the solution data to a displayable image on a graphics device, the stages being: *data enhancement, visualization mapping,* and *rendering.*

The data enhancement stage takes the raw solution data, typically defined on a mesh, and converts it to a form suitable for visualization. This frequently involves constructing a continuous model of the underlying field using an interpolation process. This interpolation process can be regarded as *filling out* the solution data with sufficient information; in the case of data obtained from scanners and remote sensing equipment however, this phase is more one of *data reduction* - the data is often collected to finer detail than the eventual display resolution.

The visualization mapping stage involves the selection of an appropriate visualization abstraction: for example, a 2D scalar field can be displayed as a contour map or a surface view. The choice of abstraction needs to be made so as to best understand the data.

The final rendering stage takes the abstract visualization object such as a contour map, and renders it on the display surface.

This three-stage visualization pipeline is summarized in Figure 1.

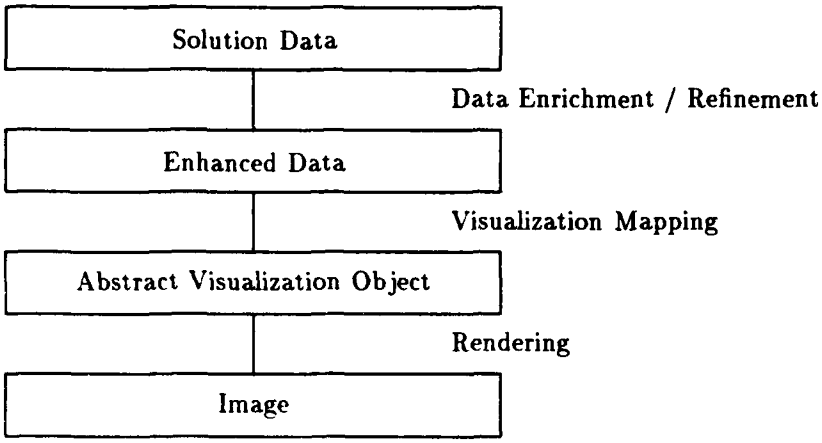


Figure 1 - Haber-McNabb Visualization Model

A similar visualization model was put forward by Upson et al (1989) as a basis for the Application Visualization System (AVS). They distinguish two cycles in the problem solving process: a *computation* cycle and an *analysis* cycle. The computation cycle consists of the following steps:

1. A theoretical stage, where the appropriate physical laws are determined.
2. A programming stage, where the physical laws are transformed to a computer program.
3. A specification stage, where details such as the computational grid, boundary conditions, etc are defined - these are essentially parameters of the computer program.
4. A computation stage, where the program is executed under the specified conditions.
5. An analysis stage where the results are examined and either deemed acceptable, or suggest further work. The cycle may be resumed from any of the first three steps, depending on the analysis of what is required.

The analysis stage is itself a cycle of three steps, roughly equivalent to the three-stage visualization pipeline of Haber and McNabb. The model is illustrated in Figure 2.

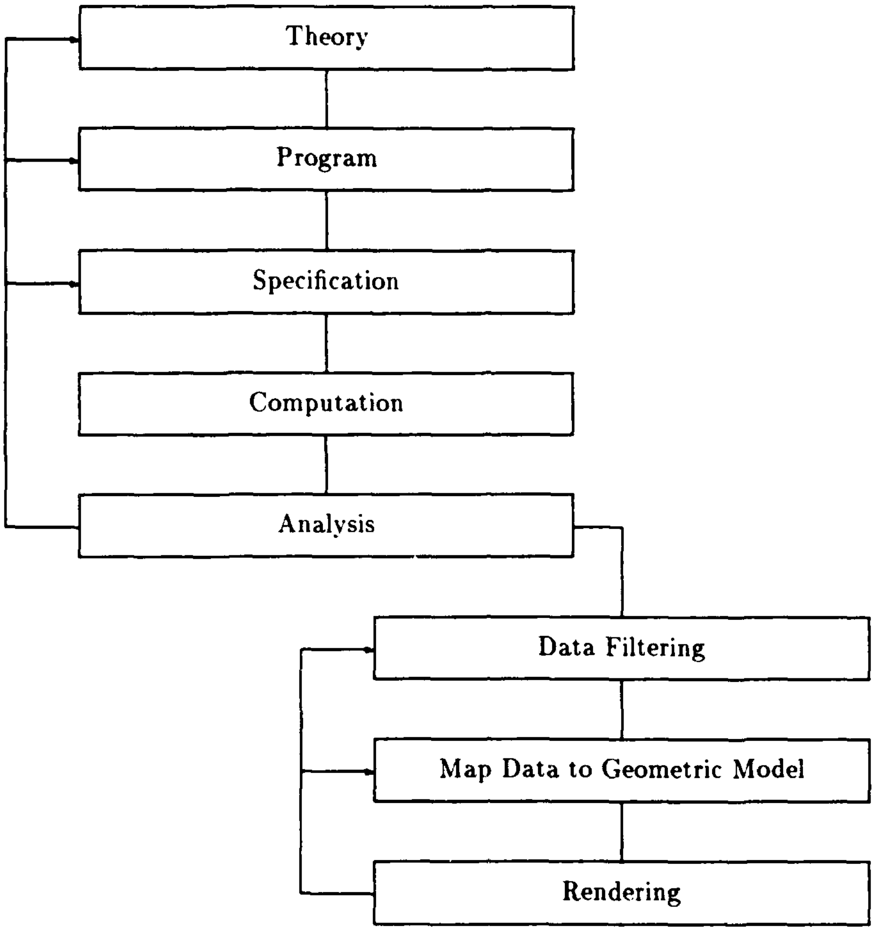


Figure 2 - AVS Visualization Model

Another view of the 'computational' and 'analysis' cycles is given in the paper by Carpenter (1990).

2.3 Classification of Techniques

With the variety of types of data and techniques available, it is useful to have a classification scheme to place some order on the subject. A number of schemes have been proposed. For example, the Bergeron and Grinstein (1989) scheme is based on the data that is to be visualized. They introduce the concept of *lattices* of data. A p-dimensional lattice of q-dimensional data is written L^p_q. The dimension of the lattice indicates the ordering of the data - zero-dimensional is unordered (eg a set of points), one-dimensional is a vector of data elements (eg a list of points to be connected by a polyline), and two-dimensional indicates an array of data elements (eg height and pressure at the nodes of a rectangular grid). The dimension of the data refers to the number of components in a data element. Thus a list of points in 3D is a one-dimensional lattice of three-dimensional data, an object of type L^1_3.

Bergeron and Grinstein go on to describe the visualization process as a sequence of transformations on lattices. The process of interpolating scattered data on to a rectangular grid, for example, would convert a lattice L^0_2 to a lattice L^2_2.

Another classification scheme was developed at a UK workshop on scientific visualization (Brodlie, 1991). This is based, not on the properties of the sampled data, but on the properties of the entity, or field, underlying the data. This is the empirical model created in the 'data enrichment' stage of the Haber and McNabb pipeline. The entity is written as E, with a superscript to indicate the nature of the field, which can be Point (P), Scalar (S), Vector (V) or Tensor (T). In the case of vector and tensor fields, the dimension is indicated as a subscript; so E^{V_3} indicates a 3D vector field.

The other element of the classification is the dimension of the domain over which the entity is defined. Thus a scalar field in 2D is written E^2_2, a 3D vector field over 3D space as $E^{V_3}_3$. Further distinction is made according to the nature of the domain: if the entity is defined over ranges of values (as in a histogram or choropleth map), the subscript is written as [n]; if the entity is defined over an enumerated set (as in a bar chart), the subscript is written {n}.

It has been found that this classification scheme can code all the popular visualization techniques, from a 2D scatter plot (E^P_2) to a volume visualization (E^S_3), to a shaded contour map ($E^S_{[2]}$).

A third classification scheme uses the mathematics of fiber bundles (Butler and Pendley, 1989). A fiber bundle is a space constructed from a base space and a fiber space: one can imagine a fiber through each point of the base space. Essentially the base space corresponds to the independent variables, the fiber to the dependent variables; the bundle is the Cartesian product of the base and the fiber. For example, a line graph is formalised as follows. The base is the real line (x-axis) and a fiber is a real line through any point on the base; the fiber bundle is the set of all such lines. A section through a bundle is obtained by picking a point on each fiber - giving a line graph. Butler and Pendley develop a taxonomy of techniques, classifying them according to dimension of base and dimension of fibre. A nice feature of the scheme is that operators can be defined on the bundles to obtain, for example, cross-sections which are bundles of lower dimension.

2.4 Visualization Systems

A number of software systems have been developed, all following the visualization paradigm expressed by Haber and McNabb - with the data enhancement, visualization abstraction and rendering being implemented as a pipeline of processes. They are often given the generic term of 'dataflow systems', reflecting the passage of data through the pipeline; they are also referred to as 'application builders', since they provide a toolkit from which different applications can be created.

The first of these systems to make a major impact was AVS, developed by Stellar (Upson et al, 1989) - now part of Stardent. It provides a number of modules for accepting data into the system, filtering and enhancing the data, defining the abstract representation and rendering on the display. To build a particular application, the user connects modules together using a *network editor*. The resulting pipeline will typically have a data source at one end, and a display module at the other end. Networks can be dynamically reconfigured to allow a scientist to experiment with different representations. The system has an X-Window based user interface with Motif-like look and feel. An early criticism was the fact that it was restricted to Stellar workstations; but this situation has dramatically changed with announcements that it has been licensed by several major workstation vendors.

AVS is an active product: AVS3 is the latest release. This release has increased support for distributed visualization in which different modules run on different processors. This has been used for example in joint work by Cray and Stardent, to allow compute-intensive simulation on a Cray and visualization on a Stardent workstation (Curington and Coutant, 1991).

The proprietary nature of AVS has encouraged the development of further visualization systems, which offer a similar way of working but are intended to be freely available at least in the educational community. A major product is apE from Ohio State University, one of the US supercomputer centres (Dyer, 1990). This works in much the same way as AVS, the user constructing a pipeline of processes. A large number of modules are provided with the system, including a volume visualizer, and the user can incorporate external modules as with AVS. The system is built to allow distributed processing, with modules connecting *via* UNIX sockets.

Another product is Khoros from University of New Mexico (Rasure, 1991). This has its roots more in image processing but is being used for general 2D data visualization. A consortium (Khoros, 1991) has been established to distribute Khoros and carry out further research. A stated aim is to provide Khoros free of charge to any organisation *via* network access. Consortium members will have the privilege of early releases and certain rights to develop enhancements and redistribute the software.

2.5 Data Management Systems

Visualization can be seen as a navigation process through a large database. It is clearly important that the data be well organised and structured so as to make this navigation efficient.

The traditional approach of using flat sequential files for data storage is inefficient in storage and access. Commercial relational database systems such as INGRES and ORACLE have been used by some groups of scientific users, but they are largely oriented to business applications. There is a need to store and retrieve the data objects that naturally occur in scientific computation, particularly multidimensional grids. Thus an important development over the past few years has been the emergence of data management systems targetted at the particular needs of the scientific computation community.

One of the first examples of a system based on a scientific data model was NASA's Common Data Format (CDF). Another example is the Hierarchical Data Format, or HDF, developed at NCSA (NCSA, 1989). HDF defines a multi-object file format for transfer of graphical and floating-point data between different systems. It allows, for example, a grid of data to be stored together with scales, annotation, etc; slices through datasets can be extracted.

Simple Fortran and C calling interfaces are provided, and so the package provides a simple scientific data management system. Moreover, NCSA also supply some useful visualization tools which accompany HDF. The software is in the public domain.

A good review of data management for scientific visualization is given by Carpenter (1991). Another useful source is the report of a workshop at SIGGRAPH 90 (Treinish, 1991) which includes a comparison of different systems.

2.6 Conclusions

AVS, apE and Khoros represent the current state of the art in scientific visualization environments. Their view of the world is dominated by data visualization: a source of data is fed through a pipeline of processes reaching a display process at the end of the pipe. An application module can be inserted as the data source, and some interaction allowed with that module (Marshall et al, 1990), but the notion of a single linear pipeline from source to sink persists.

This style of working does not necessarily reflect the exploratory nature of mathematical modelling. Here the view of the world is dominated by the modelling processes, with visualization as a set of windows onto these processes. These windows control aspects at the different levels of the model (the physical, mathematical and discretised levels of Haber-McNabb) and similarly provide views of the solution at different levels. Since the process is an experimental one, recording of the data is essential so that experiments can be restarted from intermediate points with change of parameters.

Thus it seems to us that data management must play a key role in visualization for mathematical modelling. It is significant that the existing dataflow visualization systems have been developed separately from the data management systems described in the previous section. Can an improved system be created by merging the two ideas?

3 Prototype Demonstrator for GRASPARC Project

3.1 Aims of GRASPARC

The aim of the GRASPARC project is to explore how best to provide an integrated environment for numerical simulation and visualization. This environment should support interaction not just with the solution data, but also with the physical model, its mathematical idealisation and the numerical solution. Data management is to be given special attention so that a history of the computational process can be properly maintained.

This leads us to study a number of issues:

- Can we define a visualization reference model that clearly distinguishes the different phases in problem solving, and identifies the interaction and data objects appropriate to each phase?

- Can we build a system that offers these levels of interaction, and allows the scientist to explore interactively throughout the solution process?

- Are there applicable standards that will enhance the portability of the system?
- Can the system be implemented efficiently on modern computer architectures?

A first step in the GRASPARC project has been to build a prototype demonstrator which can help resolve some of these issues.

3.2 Demonstrator Problem

A relatively simple mathematical modelling problem was chosen for the prototype demonstrator: the motion of a particle in a potential field coupled to a heat bath (Cartling, 1987). The physical problem is Brownian motion; its mathematical idealization is the Fokker-Planck equation:

$$\frac{\partial P}{\partial t} = -v\frac{\partial P}{\partial x} + \frac{1}{m}\frac{dU}{dx}\frac{\partial P}{\partial v} + \beta\left(\frac{\partial}{\partial v}(vP) + \frac{\kappa T}{m}\frac{\partial^2 P}{\partial v^2}\right)$$

$$P = P(x, v; t)$$

where:

$P(x,v;t)$ is a probability density function, expressing the probability P that the particle has a given position x and velocity v;

$U(x)$ is the supplied potential (bistable); and

β is a damping factor (strength of coupling to the heat bath).

The numerical formulation is the method of lines, in which derivatives in the spatial variables x and v are replaced by differences, and the resulting system of ordinary differential equations solved by some numerical technique.

Given an initial distribution, the problem is to find out what happens over time. The potential defines two minima, representing product and reactant states separated by a barrier, across which thermally activated transitions take place. The damping factor β determines the motion of the particle: for weak damping (small β) the motion is deterministic, and for strong damping (large β) stochastic. An aim is to ascertain the behaviour of the solution P as β varies.

The scientist will want to interact at a number of levels - for example:

- to modify the mathematical idealization by changing β
- to control the numerical solution by changing the spatial discretization

To achieve this interaction, the visualization of the solution should be presented at the same level. For control of the numerical solution, the results on the computational grid should be highlighted; for control of the mathematical idealization, the corresponding approximation to the continuous solution is required.

For any given solution data set, the scientist will also wish to have different views of the data - a further mode of interaction.

Finally the scientist will wish to compare runs with different values of β, and different discretisations, and so data from each computational run must be stored. This gives effectively a tree structure containing the computational history.

3.3 Building the Demonstrator

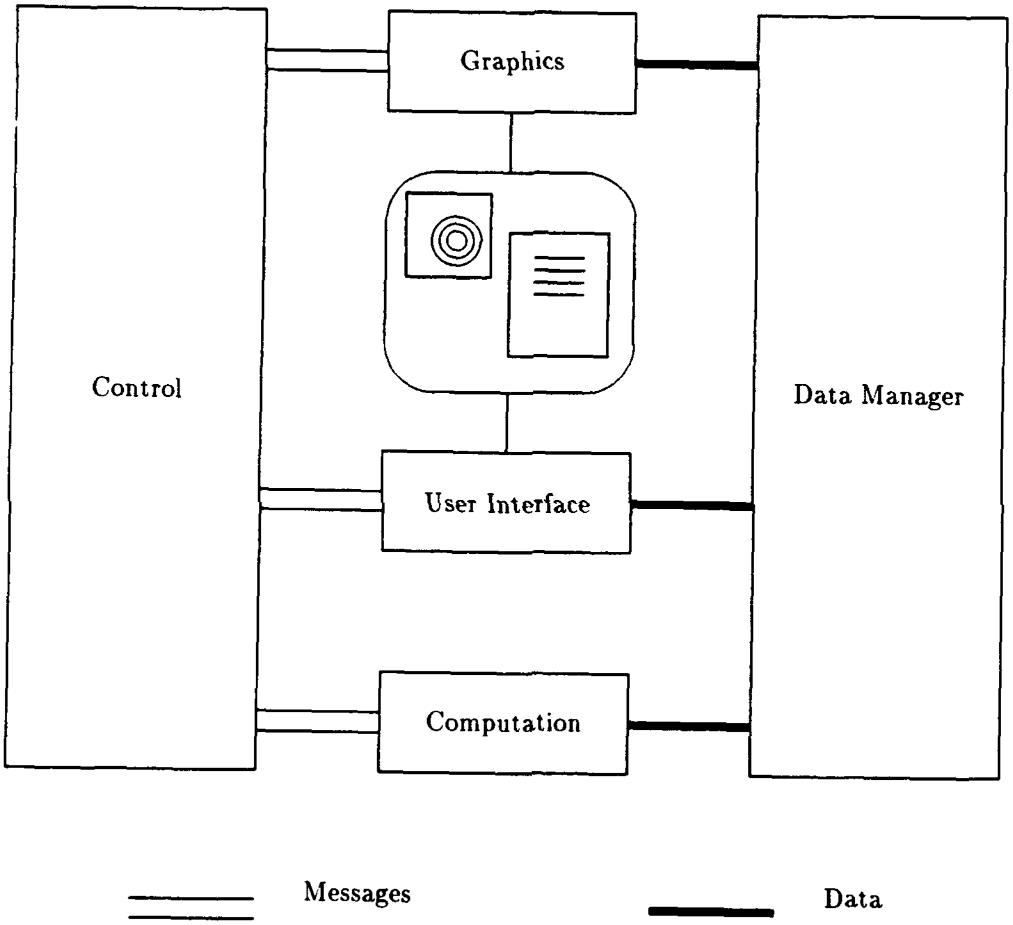


Figure 3 - GRASPARC Prototype Demonstrator

The prototype which has been built with these aims in mind has a structure as shown in Figure 3. Three separate modules implement computation, graphics and user interface components. The modules operate concurrently and communicate with each other by a message passing mechanism. Problem specification and solution data are held in a data store.

The system has been implemented on a Silicon Graphics 4D/240 workstation, with display management under control of the X Window System. Processes communicate *via* UNIX sockets, and so may run on different machines.

The computation module is based on the SPRINT software (Berzins et al, 1989) for solution of differential equations, developed at the University of Leeds in conjunction with Shell Research Ltd, and now available through the NAG Library. At times requested by the user, results are written to the data manager which is based on the HDF system described earlier. Calls to HDF are included directly in the computation module.

The graphics module consists of software to display scalar fields of two variables - contour plots and surface views - over a sequence of times. This software is written in terms of PHIGS PLUS (ISO, 1990), and allows some simple direct interaction on the part of the scientist, such as rotation of the surface.

The user interface module is based on a simple X toolkit called SUIT (Bowers and Brodlie, 1991), which allows the building of form-type interfaces. One form allows the scientist to enter control details for the computation; another, control details for the graphics.

The communications channel is implemented as a UNIX control process with socket connections to the computation, graphics and user interface modules. This control process acts as a switch for messages passed between the modules.

The system runs in the following way. On start-up, the user completes a form specifying details of the computation: mesh-size for discretisation, times at which output is to be reported, and so on. As the computation proceeds, messages are sent at each time step to the user interface module which draws a computation tree showing the progress. This acts as a *monitor window* on the computation. Immediately results are written to the data manager, the scientist may call up the graphics module to display the solution. The computation can be halted at any time, and restarted with different parameters. For example, the scientist can restart the computation with a different start time and change the time interval between time steps at which output is written. This is then displayed in the monitor window as a new branch on the tree. Figure 4 illustrates a series of such branches which together form a history tree structure. The scientist is able to trace the progress of the experiment easily.

4 Conclusions and Further Development

The prototype has achieved its aim of teasing out some of the difficult issues in developing a visual environment for mathematical modelling.

It has provided a basic understanding which is now allowing us to develop a reference model for problem solving, in which the physical, mathematical (or functional) and numerical layers are cleanly distinguished.

It has highlighted the importance of a data store as an intermediary between computation and visualization, with the scientist separately controlling the two processes. The interface to the data store becomes a key part of the system. Currently this is implemented directly in terms of HDF. However HDF does not directly support the computation tree structure that emerges in exploratory working, and so a small interface layer has been designed above HDF. This gives an 'Application Programming Interface' to the data management functions which supports the needs of a GRASPARC-like system. This will have the added advantage of removing the direct dependency on HDF, and allowing the introduction of any standard scientific data format, should that appear in the future.

The graphics module already makes use of a (draft) international standard, namely PHIGS PLUS. The prototype includes just two simple techniques, but the graphics module will eventually contain a range of techniques for displaying the various categories of fields described in section 2. The use of PEX (Gaskins, 1991), an extension of X to support PHIGS PLUS, will give the correct separation between graphics and display. This will allow the inclusion of specialist hardware to act as a PEX server - this to be built by Quintek Ltd as part of the project.

The user interface module will progress to the use of an established *de facto* standard toolkit such as Motif (OSF, 1989).

However a major study required will be the form of interaction which is most useful to the scientist. Some of the issues to be resolved are:

- What level of programmability is required? For example, current visualization systems use the concept of a network editor to dynamically configure the visualization pipeline. Can this idea be extended to the different style of working in GRASPARC?
- What control can usefully be given to the scientist over the numerical solution process - do adaptive algorithms largely remove this need anyway?

The ultimate test of the GRASPARC system will be its usability to the scientist.

Acknowledgements

GRASPARC is a collaborative project funded jointly by the UK Department of Trade and Industry and the Science and Engineering Research Council. The partners are NAG Ltd, University of Leeds and Quintek Ltd.

Our thanks go to all who are contributing to the project, especially Steve Hague, Lesley Carpenter, Richard Brankin, Greg Banecki and Alan Gay of NAG Ltd; and Pat Mills and Mark Powell of Quintek Ltd. Thanks also go to Neil Bowers, an SERC/NAG CASE award student at Leeds University who has developed the SUIT software, and to Justin Ware who assisted with the computation software.

References

R.D. Bergeron and G.G. Grinstein, 1989, *A reference model for visualization of multidimensional data*, Eurographics '89 Proceedings, Elsevier Science Publishers BV, p393-399.

M. Berzins, P.M. Dew and R.M. Furzeland, 1989, *Developing software for time-dependent problems using the method of lines and differential-algebraic integrators*, Applied Numerical Mathematics 5, pp375-397.

N. Bowers and K.W. Brodlie, 1991, *SUIT: A Portable User Interface Toolkit*, in preparation.

K W Brodlie, P M Dew and M Berzins, 1991, *GRASPARC: A Visual Environment for Numerical Computing*, NAGUA News, Issue 5.

K.W. Brodlie, 1991, editor *Visualization Techniques*, in *Scientific Visualization - Techniques and Applications*, eds Brodlie et al, Springer-Verlag, to appear.

K W Brodlie, L A Carpenter, R A Earnshaw, J R Gallop, R Hubbold, A M Mumford, C D Osland, P Quarendon, eds, 1991, *Scientific Visualization - Techniques and Applications*, Springer Verlag, to appear.

D M Butler and M H Pendley, 1989, *A Visualization Model based on the Mathematics of Fiber Bundles*, Computers in Physics, Vol 3, Number 5, pp45-51.

L A Carpenter, 1990, *The Visualization of Numerical Computation*, in Proceedings of Eurographics Workshop on Scientific Visualization, Clamart.

L.A. Carpenter, 1991, editor *Data Management*, in *Scientific Visualization - Techniques and Applications*, eds Brodlie et al, Springer-Verlag, to appear.

B. Cartling, 1987, *Kinetics of activated processes from non-stationary solutions of the Fokker-Planck equation for a bistable potential*, J. Chem. Phys. 87 (5).

I.J. Curington and M.D. Coutant, 1991, *AVS - A flexible interactive distributed environment for scientific visualization applications*, Proceedings of 2nd Eurographics Workshop on Scientific Visualization, Delft, to appear.

D.S. Dyer, 1990, *A Dataflow Toolkit for Visualization*, IEEE Computer Graphics and Applications, Vol 10 (4), pp60-69.

T. Gaskins, 1991, *PEX Sample Implementation API Overview*, Proceedings of 5th Annual X Technical Conference.

GHOST, 1982, *GHOST80 User Manual*, UKAEA Culham Laboratory.

R.B. Haber and D.A. McNabb, 1990, *Visualization Idioms: A Conceptual Model for Scientific Visualization Systems*, in *Visualization in Scientific Computing*, eds G.M. Nielson, B. Shriver and L.J. Rosenblum, pp74-93, IEEE.

ISO/IEC, 1988, *Information processing systems - Computer graphics - Programmer's Hierarchical Interactive Graphics System - Part 1 - functional description*, ISO/IEC 9592-1.

ISO/IEC, 1990, *Information processing systems - Computer graphics - Part 4: Plus Lumiere Und Surfaces (PHIGS PLUS)*, ISO/IEC DP 9592-4.

K. R. Jolls and D.C. Coy, 1990, *The Art of Thermodynamics*, IRIS UNiverse: The Magazine of Visual Processing, No 12, pp31-36.

Khoros Group,, 1991, *The Khoros Consortium Description*, Department of ECE, University of New Mexico, Albuquerque, USA.

R Marshall, J Kempf, S Dyer and C Yen, 1990, *Visualization Methods and Simulation Steering for a 3D Turbulence Model of Lake Erie*, Computer Graphics, Vol 24, Number 2, pp89-97.

B.H. McCormick et al, 1987, *Visualisation in Scientific Computing*, Computer Graphics, 21 (6).

NAG, 1989, *NAG Graphics Library Handbook - Mark 3*, 1st edition, NAG Ltd.

NCSA, 1989, *NCSA, HDF Calling Interfaces and Utilities*, NCSA HDF Version 3.1, National Center for Supercomputing Applications (NCSA) at the University of Illinois Urbana-Champaign.

G M Nielson, B Shriver and L J Rosenblum, eds, 1990, *Visualization in Scientific Computing*, IEEE Computer Society Press.

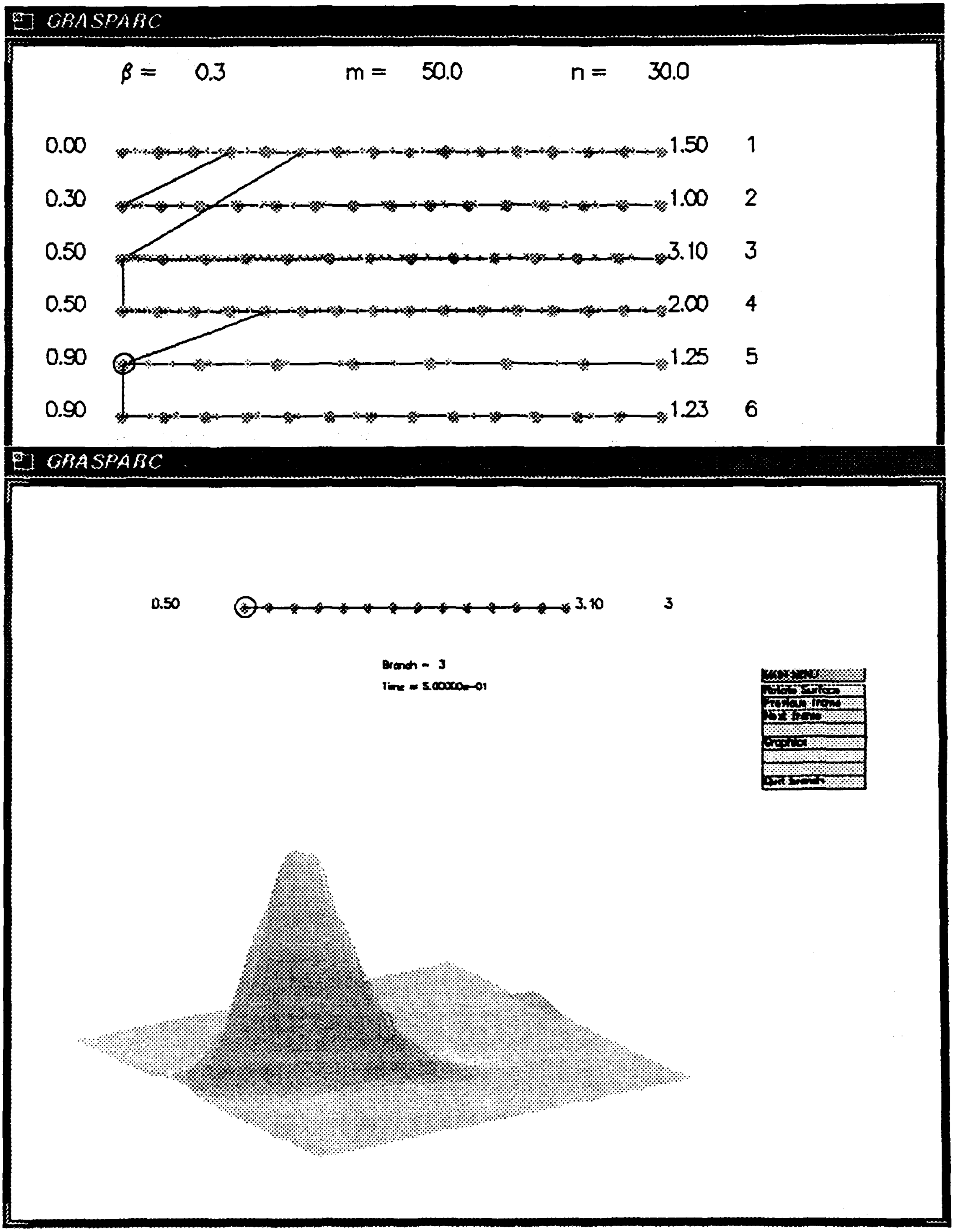
Open Software Foundation, 1989, *OSF/MOTIF Manual*, Open Software Foundation.

J. Rasure et al, 1991, *A Visual Language and Software Development Environment for Image Processing*, International Journal of Imaging Systems and Technology.

L A Treinish, 1991, *SIGGRAPH 90 Workshop Report - Data Structures and Access Software for Scientific Visualization*, Computer Graphics, Vol 25, Number 2, pp104-118.

C. Upson et al, 1989, *The Application Visualization System: A Computational Environment for Scientific Visualization*, IEEE Computer Graphics and Applications, Vol 9 (4), pp30-42.

T. West, 1991, *In the Mind's Eye*, Prometheus Books, New York.



$\beta =$ 0.3 m = 50.0 n = 30.0

Branch = 3

Time = 5.0000e-01

Figure 4

The ESPRIT Project FOCUS

C. W. Cryer*

September 2, 1991

Abstract

FOCUS (Front-ends to Open and Closed User Systems) is an ESPRIT project (European Strategic Programme for Research and Development in Information Technology) with seven industrial and academic partners.

There is a growing demand for KBFEs (Knowledge-Based Front Ends) which simplify and support the use of complex user systems. The project's main objective is to develop tools, techniques and methodologies for the construction of KBFEs for use in conjunction with "open" computational systems (meaning libraries of algorithms with potentially many applications, e.g. the NAG library) and "closed" systems (meaning application packages relating to specific problem domains, e.g. statistical packages).

The project aims at the same time to provide the means of producing KBFEs and plans to start the industrialization of such systems. It is also developing a harness which provides the framework for the interaction between end users, the front-ends and the user system (i.e. the libraries or application packages) which is considered as the back-end. The combination of the framework and its KBFE components will facilitate the provision of an enhanced human-computer interface and will also potentially provide a more profound level of assistance than is generally available at present.

*Institut für Numerische und Instrumentelle Mathematik, Westfälische Wilhelms-Universität, Einsteinstraße 62, D-4400 Münster, Germany

1 The FOCUS Consortium

The FOCUS project began in December 1988 with four industrial partners, four university partners, and one associate industrial partner. After two years, two partners dropped out owing to difficulties unconnected with FOCUS. The project is being continued in the third and fourth years by a consortium of seven partners in five European countries.

NAG	Numerical Algorithms Group Ltd, Oxford, UK (Coordinator)
IC	Imperial College of Science, Technology & Medicine, London, UK (Departments of Computing and Mathematics)
LUTCHI	Loughborough University, Loughborough, UK (LUTCHI Group)
InDeCon	InDeCon Advanced Technology Inc., Athens, Greece
Solvay	Solvay S.A., Brussels, Belgium
UPC	Universitat Politecnica de Catalunya, Barcelona, Spain
WWU	Westfälische Wilhelms-Universität, Münster, West Germany

The total project involves 56 person years of effort over a four year period at a total cost of 8 million ECUs.

2 Objectives

There is a wide and ever-growing range of application packages and libraries now available to assist computer users in a huge variety of computing tasks covering many subject areas. Some of these users write computer programs in the traditional sense, and so are mainly interested in incorporating computational units (e.g. from libraries) in their programs, but an increasing number of users program in the higher-level languages of their chosen application package(s) rather than in a general-purpose programming language. Using the term "user system" to refer to both packages and libraries, it can be seen that computer users rightly expect such systems to possess a number of qualities such as reliability, efficiency, flexibility, etc; there is also, however, a discernible, growing demand for ease of use, particularly in the initial learning stages, and for continuing support in use as the users gain confidence and tackle more complex topics within their chosen systems. It is considered that the most effective way to provide such assistance is through KBFEs (Knowledge-Based Front-Ends) which contain explicitly represented knowledge of the user system and its host environment. Existing user systems represent an enormous body of very complex and valuable knowledge that is becoming increasingly difficult to access. End users of these systems have to cope simultaneously with the intricacies of the software and with the increasing complexity of the application domain problems. For these systems, KBFEs can provide co-operative assistance to users, enabling them to use the systems more successfully. At the same time the existing know-how in the libraries and packages is preserved and their working life extended.

The two types of user systems mentioned above differ in that application packages, such as statistical packages, relate to specific problem domains and may thus be called "closed" systems, whereas libraries of algorithms, such as the NAG library, have potentially many applications and may thus be called "open" systems; hence the acronym FOCUS - Front-ends for Open and Closed User Systems.

Members of the consortium had considerable previous experience of constructing front ends for both open and closed user systems. The primary objective of the project was, and remains, to build upon this experience and to develop KBS (Knowledge-Based Systems) and HCI (Human-Computer Interface) tools, techniques, and methodologies for the construction of KBFEs for use with both open and closed user systems. As is customary in ESPRIT projects, this objective was formulated in terms of "deliverables", prescribed technological goals, to be completed and delivered to the European Commission by specified dates. The combination of industrial and university partners ensured the subsequent industrial and commercial exploitation of the results of the project.

While the initial project objectives remain, it is now perceived that the results of the project will be much more widely applicable than originally foreseen. Firstly, open and closed user systems, as defined above, are merely two points in a broad spectrum of user systems, to all of which the FOCUS technology will be applicable. Secondly, the original applications were from scientific areas, but the FOCUS technology has a potentially much broader range of possible applications.

3 Summary of the FOCUS Work Programme

The FOCUS Work Programme is divided into seven Workpackages, one of which, Workpackage 0, is concerned with organizational matters. The remaining six Workpackages, which make up the technical elements of the programme, are associated with particular areas of research and development, and are divided into several distinct tasks. They can be summarized as follows:

Workpackage 1 Distribution and evaluation within the project of KBFEs constructed by consortium members before the FOCUS project began.

Workpackage 2 Definition, implementation, documentation and development of generic toolkits for the construction and support of KBFEs.

Workpackage 3 Application of experience and tools from Workpackages 1 and 2 to the development of alternative KBFE products based on closed user systems (eg packages).

Workpackage 4 Research and development into the application of KBFE techniques to open user systems (e.g. libraries).

Workpackage 5 Development of a generic front-end harness within which the systems can operate with a high degree of independence with respect to both their operating environment and the kind of human-computer interaction (HCI) techniques being adopted by the user.

Workpackage 6 Evaluation of user requirements and practices including the development of appropriate methodologies for studying the activities of users and the effects that KBFE systems have on their performance. A database of generic tests and background results is being built up as part of this activity.

4 General Strategy

The workpackages outlined in the previous section summarize the topics covered by the work programme. However, in all phases of the project, user evaluations take place and feedback is given to the developers of the KBFE prototypes and tools. This feedback in turn motivates further development of the toolkits and KBFEs. Hence, an iterative process has been set up, based largely on the needs of the users (tool developers, tool users and KBFE users) at each stage. This approach is designed to allow the evaluation and validation of the technology being developed in the application environment for which it was intended. Methodologies for these evaluations and feedback mechanisms have been developed as an integral part of the project through Workpackage 6.

As the prototype tools and systems stabilize, they undergo additional development to bring them towards an exploitable state; commercial exploitation of the software is to begin during the project. The KBFE implementations incorporate advanced human-computer interface techniques aimed at enhancing user interfaces for a broad range of computing environments, thereby improving the portability of the systems as well as making them easier to use.

Two further characteristics of the project deserve special mention. Firstly, in order to achieve the desired flexibility, it was recognized from the start that the project should have a unifying architecture. The FOCUS architecture, a central theme of the project, is discussed in detail below.

Secondly, the development of the KBs is open-ended, in the sense that there is no single restrictive concept of what is allowed, other than that the different components must be compatible with the architecture (a very weak restriction). This allows the development of different types of KBs and greatly enhances the exploitability of the resulting products.

5 The FOCUS Architecture

Any system built using FOCUS tools will usually cater for:

1. The End User
2. One or more Back-Ends (BEs)
3. One or more Knowledge-Based Modules (KBMs).

These must communicate with one another, and the communication links must satisfy several conditions:

1. Communications with the end user must be user-friendly.
2. The communication protocols must be straightforward.
3. So far as possible, the system must be environment - independent.

The FOCUS architecture is evolving to meet these requirements, and its current form is shown in Figure 1.

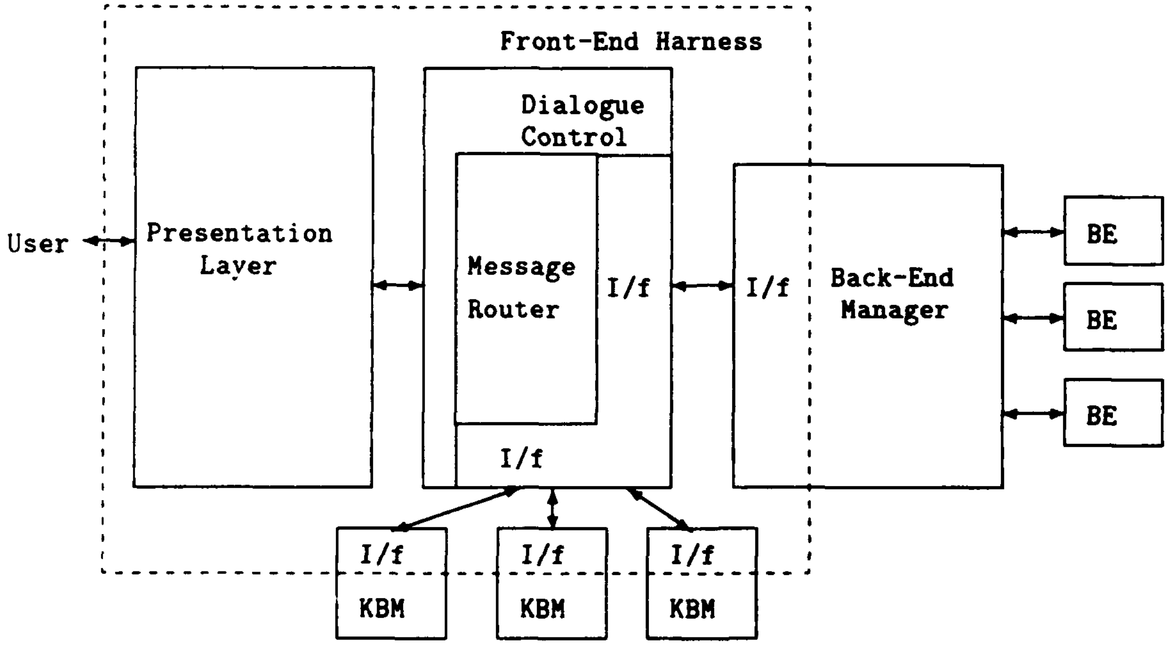


Figure 1: The Focus Architecture

The Front-End Harness (FEH) provides a versatile, portable framework encompassing interaction between the end user, the KBMs and the Back-End Manager (BEM) (which manages interaction with one or more BEs). The communications between the BEM and KBMs use a FOCUS Protocol. The design allows the components to run on separate machines with an appropriate link.

The FEH is written partly in C and partly in PROLOG, and the Presentation Layer uses the "look and feel" of OSF/Motif. However, because of the modular construction it is possible to rewrite the Presentation Layer in OPENLOOK, for example. Indeed, the use of the eXtensible Virtual Toolkit (XVT) is being investigated. This may allow the interface styles of a wide range of host systems to be adopted.

The BEM is written also partly in C and partly in PROLOG. While the FEH and the KBMs are concerned with the "whys" and "wherefores" of what the end user is or should be doing, it is the BEM that take cares of the "how", in order to maintain the concept of interface separability within this architecture.

Version 1.0 of the FEH along with version 3.0 of the BEM are being used by the most recent KBFEs, and evaluation of these latest versions is underway. The FEH is recognised as the major component of the architecture and so its development (in line with user requirements) has been accelerated. Work in the first six months of the third year will concentrate on the specification, design and implementation of version 2.0 of the FEH, building on the experiences of the KBFE developers.

Although the components of the architecture have been developed on Sun systems under Unix, the architecture is equally applicable to other environments. This has been shown by the development of some PC demonstrations running under MS/DOS and Windows-3.

6 KBFEs under Development

KBFE Prototypes are being developed to act as test-beds for the architecture, tools and evaluation strategies already developed. Moreover, the KBFEs chosen for development are "real-world" and diverse. As such these KBFEs have good exploitation potential. Some of the main KBFEs developed, or under development, will now be described briefly; somewhat longer description of two KBFEs then follow.

Routine Selector This KBFE has been developed to select an appropriate routine from a library of routines given relevant knowledge by the user. The library chosen was the NAG Fortran Library of nearly 1000 numerical and statistical routines. A working prototype exists which not only selects routines, but provides help with that selection and (sub)program generation. Currently, routine selection is not available for the whole library, but is being expanded and further higher-level interfaces for selected areas of the library are being investigated.

FAST This KBFE can be regarded as the successor to GLIMPSE (a statistical KBFE which was one of the state-of-the-art systems used by the project for initial investigations.) A portion of GLIMPSE has been re-implemented and restructured within the FOCUS architecture to produce FAST, a much more modular and user-friendly system. These improvements were a direct consequence of using the FOCUS architecture. FAST has already proved to be a good test of the architecture and will be further developed towards the full functionality of the GLIMPSE system.

SEPSOL This KBFE is aimed at helping experimenters (mainly chemists) to design their experiments. The structure of SEPSOL is based on the methodology used at Solvay to solve experimental design problems. It is organised as a sequential set of activities: Data Acquisition - Design Choice - Design Generation - Design Comparision - Result presentation. A prototype is now running.

Meradis This KBFE is aimed at helping the safety engineer of a factory in simulating the dispersion of a toxic gas after an accidental release. The knowledge elicitation is currently in progress and will be followed by the implementation of a prototype using the FOCUS architecture.

DOX Expert System for experimental design The system helps engineers in the design and analysis of factorial and fractional factorial designs with factors at two levels using economic and/or technical restrictions. It is equipped with help facilities and is expected to be an alternative to Taguchi methods.

KAFTS Time Series Forecasting This expert system is able to model time series, interacting with the user in areas where he/she has knowledge. It also provides explanation in terms of implementation details (e.g. data points required) but not in terms of statistical strategy. It provides context sensitive help.

REGS Regression expert system The system evaluates the square metre value of real state objects as a function of several explanatory variables. It is part of a more general system and performs "automatic" selection of the best regression model with minimum user interaction. It is integrated with data management capabilities and runs in a PC environment.

KBFE for Stochastic Processes This system will test the re-usability of components of the Routine Selector. A library of programs has been identified and a KBFE is to be built using many of the components from the Routine Selector.

All these KBFEs have been, or are being, built within the FOCUS architecture using FOCUS tools, but with varied back-ends and domains of application.

7 The Routine Selector

The Routine Selector is a KBM for assisting users of the NAG FORTRAN Library to select and use one of the library routines (at present almost one thousand in number). For this application, a forward-chaining rule-based expert system is suitable, as is shown in Figure 2.

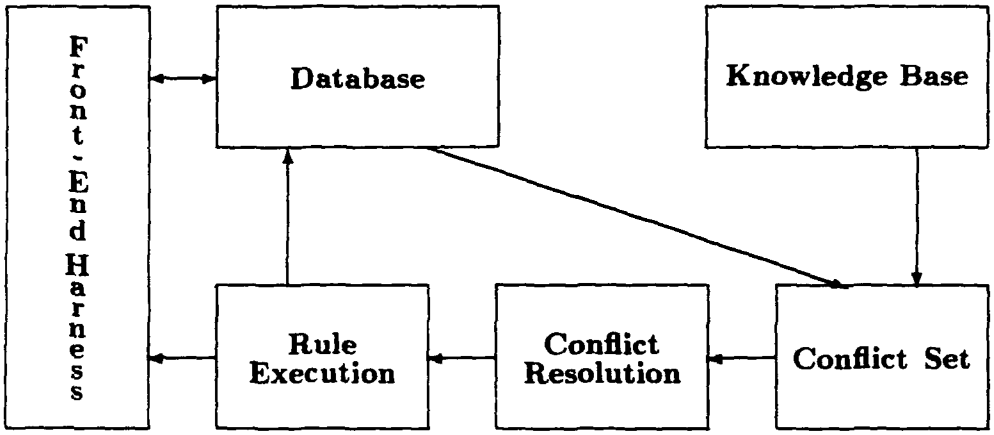


Figure 2: The Routine Selector KBM

The system was designed to build on the experience gained during implementation of the NAXPERT and KASTLE systems and in particular the following points have been addressed:

- **Truth Maintenance** At any time the user is able to alter previously entered data without compromising the consistency of the system output.
- **Interface Mode** The user is able to switch at any time between free text input of keywords and interaction with menus of suggested input.
- **Context Sensitive Keywords** The meaning of keywords entered by the user depends on the context of the current problem.
- **Keyword Abbreviation** Free text abbreviations of keywords are allowed as input.
- **Explanation** How, why and why not explanations are accommodated.
- **Uncertainty** is handled both in the specification of the knowledge base and in the data input by the user.

8 SEPSOL

SEPSOL is a knowledge-based system for experimental design developed by SOLVAY. The purpose of SEPSOL is to replace the statistician in his role of intermediary between the experimenter and the set of back-ends available to build and analyze experimental designs.

The various stages in the design of an experiment are a sequential set of activities. These activities are the basis of the structure of SEPSOL : the user is invited to execute, in turn, each of the activities to design his experiment. Five activities can be identified:

1. **Data acquisition** is concerned with eliciting as much information as possible about the experimenter's problem: description of the objectives, the factors, the model, etc.
2. **Design choice** analyzes the data entered by the user, finds the list of possible types of designs adapted to the given problem, and chooses their sizes and parameters.
3. **Design generation** is concerned with generation of the possible designs using appropriate Back-Ends.
4. **Design comparison** analyzes the different generated designs according to a set of qualitative and quantitative criteria. It checks also which of the possible designs correspond to the objectives of the experimenter.
5. **Result presentation** presents the possible designs and their relative advantages to the experimenter. It helps the user to choose the design best adapted to his problem.

Like the Routine Selector, SEPSOL has a KBM which controls the selection of the experimental design. Like the Routine Selector, SEPSOL also interacts with BEs, as shown in Figure 3.

9 Further Information

Current progress in the project is described in several public documents, e.g.:

Steve Hague and Ian Reid: FOCUS Second Annual Report: Summary of Progress.
Document reference: FOCUS/NAG/8/8.2-P, January 1991.

E. Edmonds, E. McCaid, S.J. Hague and I. Reid: The FOCUS Project: A Separable Architecture for KBFEs.
Document reference: FOCUS/NAG/6/7.2-P, August 1990.

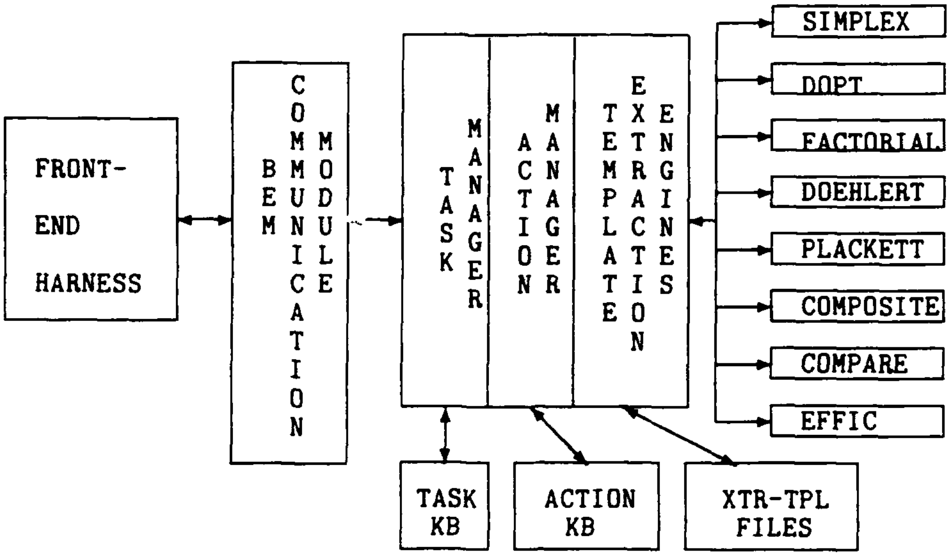


Figure 3: SEPSOL Back Ends and Back End Manager

Further details of the project may be obtained from any partner or by contacting the project officials at the coordinating site:

Dr. Brian Ford, OBE	- Project Chairman
Dr. Steve Hague	- Project Technical Coordinator
Mr Jimmy Brown, CB	- Project Administrator

Address:
NAG Ltd
Wilkinson House
Jordan Hill Road
Oxford OX2 8DR
UK

International Phone: +44 865 511245

International facsimile: +44 865 311205

Network: (JANET)
- BRIAN @ UK.CO.NAG (Dr. Ford)
STEVE @ UK.CO.NAG (Dr. Hague)
JIMMY @ UK.CO.NAG (Mr Brown)

10 Acknowledgements

The work of the FOCUS project is supported in part by the European Commission under its ESPRIT II initiative. Members of the FOCUS Consortium gratefully acknowledge the funding support provided.

Using FOCUS Technology to Build Front Ends

*S. J. Hague**

July, 1991

Abstract

FOCUS (Front-ends to Open and Closed User Systems) is an ESPRIT project, the purpose of which is to provide tools, techniques and methodologies for the construction of knowledge based front-ends (KBFEs). The paper is intended to be read in conjunction with other reports of FOCUS work elsewhere in the conference proceedings. The particular theme of this paper is the manner in which several, diverse KBFE prototype exercises are being documented; a description of a particular exercise is then given in greater detail.

*The Numerical Algorithms Group Limited,
Wilkinson House, Jordan Hill Road,
Oxford, OX2 8DR, United Kingdom.

1 Introduction

This paper reports on an activity to develop useful technology which facilitates the construction of front-end systems; that is, systems which serve as an interface between the end user and one or more so-called "back-end" systems. Typically the back-end systems are computational software packages or collections of library subprograms for solving certain application problems. If the front-end system incorporates reasoning components, than it may be referred to as a knowledge based front-end. The KBFE technology activity in question is being undertaken on a collective basis within a European Community ESPRIT project called FOCUS. A description of the project, in terms of its structure, aims and general technical strategy, appears elsewhere in the proceedings of this conference. In this paper we will concentrate on the application of FOCUS technology in building a prototypical KBFE. We refer both to the general approach adopted for KBFE construction, and to a specific construction exercise, that happens to be an application of time series analysis in the commercial sector. The description provided here is intended to be complementary to other reports on FOCUS work given elsewhere in these proceedings; each report is concentrating on some specific aspect of the project. There is, however, a common underlying theme, namely the search for generic technology which, it is hoped, can be successfully applied in diverse, practical circumstances where existing applications software can usefully be supplemented by front-end HCI and reasoning components.

In section 2 of this paper, we provide a brief overview of the FOCUS architecture; that is, the framework provided by FOCUS for the integration of HCI components, one or more back-end systems, and optionally, knowledge-based modules which provide reasoning capabilities. Sections 3 and 4 are devoted to the descriptive form ("template") devised within FOCUS to ensure that the several prototypical KBFE exercises being developed as test-beds for the FOCUS technology are undertaken in a generally consistent manner. The next section then presents an instance of the template, composed for a particular KBFE exercise. Finally, we present some general observations and conclusions in section 6.

2 The FOCUS Architecture – A Brief Overview

Because a more detailed description of the FOCUS architecture appears in these proceedings, we confine the description of that architecture in this paper to a brief overview in order to emphasize key points which relate to the following discussion of KBFE construction exercises.

First, we should clarify what is meant by the term "architecture" in the context of a project such as FOCUS. The purpose of the project is to develop tools, techniques and methodologies for the construction of (knowledge-based) Front-Ends (FEs) existing software libraries and packages, the so-called Back-Ends in FOCUS terminology. The term "architecture" in the FOCUS sense is not intended to suggest a rigid and detailed internal design of a specific software system but instead conveys the notion of a framework

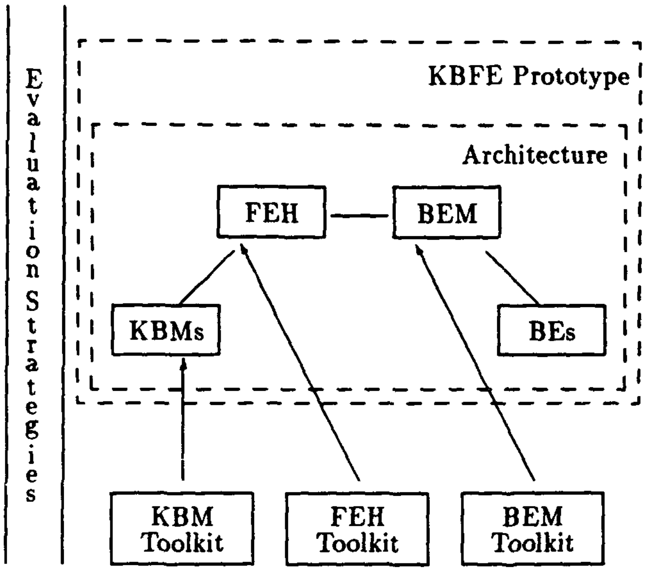


Figure 1: Integration of the FOCUS Components

for uniting major software components, some of which (typically the Back-End) exists, and other (e.g. knowledge-based components) to be developed with the aid of appropriate FOCUS toolkits. The framework (or architecture) is specific to the extent that it defines the location of those components to one another in terms of communication, and it also incorporates a protocol which governs the form of message passing between those components.

The Front-End Harness, as indicated in the diagram below, is a FOCUS-developed component which serves as a programmable HCI communications hub (both with the end user and between software components) within the architecture. The internal form of the other components (the Back-End(s) and the knowledge-based modules) is of no direct significance to the conceptual architecture, the only requirements for integration into a FOCUS framework is that those components conform directly or through interface layers to the FOCUS message passing protocol. The various components involved need not all be on the same host system so it is fair to summarise the FOCUS architecture is providing a distributed and separable framework for constructing front-end systems.

In simplified diagrammatic form the FOCUS architecture can be summarised thus:

where the terms FEH, BEM, BE, KBM stands for Front-End Harness, Back-End Manager, Back-End and Knowledge-Based Modules respectively. As the diagram indicates, the use of FEH, BEM and KBM are supported by their respective FOCUS-provided toolkits. Further details of these toolkits are available from the Project Co-ordinator upon request.

3 Describing a KBFE – The General Approach

As the paper by Colin Cryer (elsewhere in these proceedings) indicates, several KBFEs are being designed, developed and evaluated in the FOCUS project, as prototype exercises in the use of the emerging FOCUS technology. The set of prototype KBFEs chosen relate to application areas which are diverse in nature and have real-world relevance, thus providing an effective, collective test-bed for the genericity of that technology, i.e. the FOCUS tools, architecture and evaluation strategies.

To ensure consistency of treatment, each of the KBFE prototypes is described in a uniform way through the use of a descriptive template, the form of which has been developed within the FOCUS project. For each KBFE developed, in development or planned within FOCUS, a description based on the template is initiated and then periodically updated as the particular KBFE exercise progresses. The template contains seven sections, viz;

1. global problem description
2. problem definition
3. actors description
4. problem solution
5. purpose of the KBFE
6. industrial issues of the KBFE
7. contact information.

A full copy of the KBFE prototype descriptive template appears as section 4 of this paper. To amplify the above section summary a little, however, it is worth noting that the first four sections refer essentially to some industrially relevant, existing real-world problem area (in which no KBFE is presumably available). These sections must set the scene in describing the overall domain of application in question, why some problem or class of problems arise, to what extent existing Back-Ends application software addresses that problem or class of problems. They must also describe who would be involved in posing and solving the problem(s), and how a KBFE could be of use. Sections 5 and 6 deal with the design of the KBFE and its development, both within the FOCUS project and potentially in subsequent commercial exploitation, where relevant. The final section names the individuals within the project who are to serve as points of contact.

With regard to the participants in each KBFE prototype exercise, it is also worth remarking on the so-called "actors" (section 3 of the template), using their own problem-specific terminology. The term "actors" is intended to embrace primarily three groups of individuals:

- a representative sub-group of Problem Posers; these are the potential end-users of the resulting KBFE
- experts in the problem domain (the template uses the abbreviation "Exp-Dom" to refer to these)
- experts in the use of the one or more Back-Ends involved ("Exp-BE").

These primary participants or "actors" in each KBFE exercise are likely to be assisted by:

- a knowledge engineer (KE) who assists the Problem Posers and Experts to formalize their knowledge, and supervised by
- a Project Manager (PM) responsible for the general coordination of the specific KBFE exercise.

The emergence of the KBFE descriptive template, and its instantiation in the case of each KBFE prototype undertaken, is characteristic of the overall FOCUS approach, namely a search for the extent to which generic technology and methodology can be developed to support the construction of diverse and realistic front-ends. The template itself has been refined on several occasions during the lifetime of the project,as more experience has been gained in constructing KBFEs; the essence of that cummulative experience is then distilled into an updated and stabilised form of the template.

4 The KBFE Descriptive Template

The following is a largely verbatim extract form current FOCUS internal technology guidelines. It describes the generic form to which all FOCUS KBFE constructors must confirm when undertaking their exercise. The style used is that of a questionnaire in which the KBFE constructor is asked to provide information to a series of requests for information. The various sections of the questionnaire are labelled A to G here, in order to avoid confusion with the sections of this paper:

A. Global problem description

A.1

Give a description of the application domain and a description of the class of problems that the final KBFE should solve. If the KBFE is intended to deal with various classes of problems, each of these should be described separately.

Supervisor: Exp-Dom
People concerned: Problem-Poser, Exp-BE

A.2

Give the description of a specific example problem which should be solved by the final KBFE and express it in words of the Problem Poser (End User of the final KBFE). This example should be representative of the most common problems.

Supervisor: Problem-Poser

B. Problem definition

B.1

Describe the **class of problems** in terms of:

- Initial state, including the list of actors (classified into categories e.g. Problem-Poser, Expert, Back-End)
- Goal state

B.2

Illustrate the above description with a specific example.

Supervisor: Exp-Dom
People concerned: KE, Exp-BE, Problem-Poser

C. Description of the Actors

The actors involved in the problem resolution will be described here. The way in which in each actor should be described depends upon his or her role in the problem resolution. The information needed is given for each category: Problem-Posers, Experts and Back-Ends.

C.1 Description of the Problem-Posers

The purpose of this section is to describe the potential End-Users of the future KBFE. In some applications, it may be necessary to classify them in sub-categories especially when:

- their knowledge of the domain and of the BE are very dissimilar
- the KBFE is aimed to solve a large range of problems

For each class, the following information should be given to describe the potential End-Users:

- Education
- Knowledge (and lack of knowledge):
  - in their domain (physics in our example)
  - in the application domain (statistics in our example)
  - in the use and functionalities of the Back-End with the frequency of back-end usage
- Set of operations that users can perform

Supervisor: Problem-Poser
People concerned: Exp-BE, Exp-Dom, KE

C.2 Description of the Experts

For each expert whose knowledge should be included in the KBFE, the information needed is:

- Knowledge:
  - in the Problem-Posers' domain
  - in the application domain
  - in the use and functionalities of the Back-End
- Set of operation he can perform

Supervisor: KE
People concerned: Exp-Dom, Exp-BE

C.3 Description of the Back-Ends

For each Back-End:

- Type of the BE: (library, command driven program, batch program, data base ...)
- Technical description of the inputs and outputs
- List of the BE functionalities

Supervisor: Exp-BE

D. Problem Solution

D.1 General Resolution Process

Give the general resolution process for the class of problems to be solved by the final KBFE. Structure the complete problem into a sequence of subproblems and briefly explain each of them. There is no need to described the complete tree of operations. For each step of the resolution, state which actors are involved. If the KBFE is intended to solve different classes of problems, the resolution process should be given for each of them.

D.2 Illustration

Illustrate by giving the resolution process for the concrete example presented in A.1.

Supervisor: KE
People concerned: Exp-Dom. Exp-BE

E. Purpose of the KBFE

Give the list of requirements for the KBFE according to:

- the difficulties and lack of knowledge of the Problem-Poser
- role played by the expert in the resolution process
- wishes of the potential End-Users for the user interface
- characteristics of the BEs

Supervisor: KE
People concerned: Problem-Poser, Exp-dom, Exp-BE

F. Industrial issues of the KBFE

This Section is necessary to check the industrial relevancy of the applications developed within FOCUS. Its aim is to precisely describe the environment in which the KBFE will be developed and the project plan for the KBFE.

F.1 Who is going to develop the KBFE?

Name(s) of the partner(s) and of all the persons who will be involved in the development of the KBFE with their role and working experience (Expert, Knowledge engineer...)

F.2 Which tools are available for the implementation?

FOCUS tools and other tools.

F.3 On which hardware are the BE and KBFE intended to run?

F.4 Is there an industrial interest and economical payoff in developing a KBFE?

- number of potential users, expected frequency of use, strategic importance for the Company
- are there similar systems available on the market?
- is there any economical return expected?

F.5 Give your internal project plan

List the successive steps you intend to follow in the developments of the KBFE.

- How do you plan to acquire the knowledge?
- Are the BEs ready to run on the desired hardware?
- Do you plan to develop the KBFE in different stages?
- Which are the objectives of the first prototype?
- Do you have End-Users available to test the successive prototypes?
- Which are your deadlines for the differe.it steps of the development? Give a Gantt Chart.
- Give your evaluation plan as agreed with the FOCUS evaluation team and indicate its current status in the required form (as stated in the appropriate FOCUS project note).

Supervisor: Project Manager
People concerned: KE

G. Contact

Give details of a contact with whom problems can be raised.

5 Describing a KBFE – An Example

We now present an instance of a FOCUS KBFE description, composed along the general guidelines of section 4, the descriptive template. The description given here is a slightly abridged and adapted version of an actual KBFE description document; the changes made in this version relate only to the omission of certain specific details which are not directly relevant to the overall theme of this paper. The KBFE in question provides assistance in the use of univariate time series modelling and forecasting using the approach of Box and Jenkins. It is intended for use in banking and related financial applications. Further details of the KBFE can be obtained upon request to the FOCUS Project Co-ordinator. Though it serves as a worthwhile, practical example of the efforts being made by the FOCUS project to validate its emerging technology in diverse, real-world applications any one of the several other KBFE prototype building projects could also have served in an examplary role – the point is that FOCUS is endeavouring to apply generic technology in a consistent way. The first subsection below presents the global problem context for the chosen KBFE, and the subsequent subsections follow (except where abridged, as explained above) the outline described in section 4 of this paper.

5.1 Global Problem Description

5.1.1 Description

Box and Jenkins methodology for the evaluation of time dependent statistical data has several levels of sophistication: Univariate, Intervention analysis, transfer function and multivariate models. In this application, attention is focussed in the first level i.e univariate time series models.

A typical end-user may have to in obtain a forecast for a number of time series which are relevant to his or her organization. For each relevant time series, it will be necessary to fit to the data, a model belonging to the general class of Seasonal AutoRegressive Integrated Moving Average (SARIMA) models as described by Box and Jenkins. This will require and iterative strategy of model identification, model estimation and model validation or diagnostic checking.

Once a reasonable model has been obtained, it can hopefully be used to generate the desirable forecasts.

One could also consider the possibility of controlling the forecast errors by means of statistical control in order to decide when the model has to be modified.

5.1.2 Example

Suppose that a banking organization (Bank X) has 100 branches or agencies and in each agency there are 10 relevant time series to be forecasted : total client's deposits, total deposits of long-term kind, etc.

Bank X wants, once per year, to get the forecasted values of each of its one hundred branches for each of the ten categories. That means 1000 time series for the next year. Then, even if Bank X has an expert in Box and Jenkins methodology, solving the problem by using directly available statistical software will require considerable effort.

Therefore Bank X will prefer to use a KBFE, which encapsulates the expertise of an in house expert, and provides reasonable forecasts for a high percentage (if not all) of the time series.

5.2 Problem Definition

5.2.1 General Problem

Initial State:

The data:

> Probably stated in a data base with possibilities of updating, modifying, deleting, extracting by different criteria, etc.
>
> For each time series, a minimum of data equal to 3 times the seasonal in question (e.g. one year) is required.

The experts:

> The end user of the KBFE, when constructed, will probably be competent and perhaps even an expert in the domain from which the time series comes (banking, electricity consumption,etc.)
>
> Also an expert in the Box-Jenkins methodology, at least in univariate time series modelling is required.

The Back-end:

There is a number of available statistical software packages available to solve the problem, e.g.GENSTAT, SCA, X-11, X-11 ARIMA.

Goal State:

The goal is to satisfy the end user. In this case, this means presenting "good" forecasts for each time series, in a short period of time together with some measure of error, standard deviation of the predicted values, confidence intervals, etc.

The final results should be both in graphic and list form.

Also, in the case that the KBFE has not been able to find an adequate model for a particular time series, this should be brought to the attention of the end user together with the reasons why this has happened.

Now we list the operational steps (Subproblems) in achieving the above goal.

For each subproblem, its initial state, sub-goal state and required actions are identified:

Sub-problem 1: Get the data

Initial state:

The data of the time series to be modelled are stored in the database. If not tell the user to enter these time series.

Goal state:

The data are in a file.

List of actions:

- Ask the user which time series he wants to model.
- Locate the time series in the database.
- Extract the time series.
- Verify this is the requested time series.
- Store in an appropriate file.

Sub-problem 2: Explory data analysis

Initial State:

The data of the particular time series is in a file.

Goal State:

The same data after corrections for obvious inconsistencies are stored in a file (possibly the same).

List of actions:

- Present several plots to the end-user.
- Ask the user for corrections.
- Do the corrections.
- Store the data (possibly transformed).

Sub-problem 3: Model identification (Note: this phase of the KBFE requires an expertise in Box-Jenkins (BJ) methodology and in the use of one or more of the Back-Ends to be used).

Initial State:

Data and its seasonality.

Goal State:

At least one tentative SARIMA model for this time series.

List of actions:

- If the end-user is not an expert in BJ methods, then run the Back-End.
- With or without user's interaction, obtain at least one tentative SARIMA model.
- Store the necessary information that will be necessary for estimation.

Sub-problem 4: Model estimation (Note: as per sub-problem 3).

Initial State:

The model is identified in the previous subproblem and the data.

Goal State:

The estimated values of the model's parameters and relevant information for diagnostic checking (t-values of the parameters, correlation matrix of the estimates, conditioning of this matrix, roots of the AR and MA polynomials, residual mean square, the residuals, etc.).

List of actions:

- Prepare the input for the back-end.
- Run the back-end (estimation).
- Extract all relevant information.
- Store this information.

Sub-problem 5: Diagnostic checking (Notes: this requires BJ expertise)

Initial State:

The stored relevant information from above.

Goal State:

Accept or reject the model.

List of actions:

- Apply the expertise of the BJ Expert to the relevant information.
- If the model is accepted, then go to Subproblem S7.
- If the model is rejected then go to Subproblem S6.
- Limit the possible loops to a maximum of 2 or 3 interactions.

Sub-problem 6: Model re-specification

Initial State:

A model that has been rejected in Sub-problem 5, plus the residuals of the model, and the BE.

Goal State:

A new model for Sub-problem 5.

List of actions:

- Prepare input for running the BE with the residuals.
- Run the BE.
- Obtain a new model.
- Add it to the model of the series.

Sub-problem 7: Forecasting

Initial State:

The data and an acceptable SARIMA model. Also origin of forecast and number of forecast, etc.

Goal State:

The desired number of forecasted future values, with their standard deviation or confidence interval in graphic and in list form.

List of actions:

- Prepare input for the back-end.
- Run the back-end (forecast).
- Present the results to the user.

5.2.2 Specific example

In the interests of brevity, details of the specific example, as required by the generic template, have been omitted here, but are present in the actual description for this KBFE.

5.3 Description of the actors

5.3.1 Problem Posers

In the context of this KBFE exercise, the problem-posers can be of two possible types, each with different knowledge depending on the delegation policy of the bank in question.

1. Managerial Personnel e.g. Manager in charge of planning.
2. Administrators. e.g. clerk in above manager's department.

Managerial personnel presumably have a greater overall understanding of the problem domain, namely the trends in banking practices expressed as time series. However it, may be that the administrators are more familiar with the use of computers and computer software. Neither is likely to have any detailed knowledge of a specific back-end nor of the statistical application domain (time series analysis). It is assumed that the general educational level of Managerial staff is greater than that of administrative staff, although this may also depend on age.

5.3.2 Experts

It is apparent that our KBFE construction exercise needs expertise from at least three difference sources:

- a problem domain (banking) expert
- an expert in time series analysis
- an "expert", i.e. a competent user, of a computationally capable statistical back-end package.

It is probably unlikely, given the application domain in question, that a single individual will be capable of fulfilling all three roles. a more pertinent question would be whether or not the second and third of the experts listed above would or could be the same person. The second-listed person, typically a professional statistician specializing in time series analysis, may be familiar with a specific back-end package, e.g. SCA, but the bank in question may have chosen some other package, Genstat, for example. The difficulty that the statistical expert might have in translating his experience of use of one Back-end to another, should not be under-estimated. It is sensible, therefore, to postulate that there would be three distinct individuals in supplying primary sources of expertise.

A further source of expertise (apart from the use of FOCUS technology, that is) might also be required, depending on the volume and organisation of the data involved. The statistical Back-End may or may not possess its own data management system, so it would be prudent to assume that multiple Back-Ends may be involved, one of which providing a database facility. If that is so, that the involvement of a database person, who may or may not be distinct from those "experts" already identified, should be assumed.

The table below summarises the likely expertise level of the potential (human) "actors" in the KBFE exercise, using a scale of 3 to 0 for high, intermediate, low, zero expertise in the indicated areas of competence:

	Banking Principles	Time Series Analysis	Use of BE(stats)	Use of BE(db)
Bank Manager	3	1-0	0	1-0
Bank Administrator	2-1	0	1-0	1-0
Time Series Analyst	1-0	3	3-0	1-0
Back-End Expert	1-0	2-1	3	2-0
Data Base Expert	1-0	1-0	1-0	3

For example, the expert in the use of a statistical Back-End is likely to have little or no detailed appreciation of relevant banking principles, probably has some appreciation of TSA techniques, is by definition judged to be highly competent in one or more specific Back-Ends, and should have some appreciation of data base issues, though that might not be the case.

An aside: it is important to the success of any KBFE exercise that such a "matrix of competence" is recognised, for it is the KBFE, developed and supported by FOCUS technology, that is acting as an integrator of sources of competence that presumably have not been (successfully) brought together previously.

5.3.3 Description of the Back-Ends

A typical collection of Back-Ends is given below:

1. Idaut: a batch program which automatically identifies a tentative model of a time series for use in the Model building stage of the Box-Jenkins methodology. Data files:

2. Input: seasonality, time series

3. Outputs: d, D, P, p, q, Q, Q0 (Statistical parameters).

4. SCA: a general statistical package which can perform time series analysis and can be used in both bath and command mode.

   Inputs: SCA command or command file.
   Outputs: Varials.

5. Genstat: is a batch driver general statistical package.

Inputs: Genstat command program.
Outputs: Data files.

6. Data base: e.g. Oracle on Sun.

5.4 Problem Solution

See §5.2.1

5.5 Purposes of KBFE

1. To provide automatic forecasting of a given time series interacting with the user only in areas where the user is knowledgeable ie in this case:

- source of the data
- interface requirements

2. To provide explanation only in terms of implementation details such as:

> "unsufficial data points to make a forecast; accurate model could not be identified"

but not in terms of statistical strategy, as this would be meaningless to the typical end user in the banking context.

3. To provide context sensitive help-context, i.e. meaningful in terms that the end user can understand.

5.6 Industrial Issues

5.6.1 Personnel

For the KBFE construction exercise in question, three "experts" have been identified, covering five domains of competence; time series analysis, data base use, and use of three back-end packages - Genstat, SCA and Idaut. Banking expertise is being provided by sources outside the FOCUS project. Other technical expertise, in the use of the Front-End Harness and KBM, is drawn from elsewhere in the project.

5.6.2 Project Plan

The plan for this particular KBFE construction exercise deals with several specific issues, including:

- knowledge acquisition; being acquired by the interview method
- choice of Back-Ends both in terms of functionality offered, and the host systems on which they are available
- KBFE development; to be undertaken in stages, with the first version being produced within four month, an evaluation period of two and half months, followed by a second, enhanced version, intended to be the "deliverable" required by the project's contractual commitments. That second version was planned to last three months, with a further month for additional refinement. Thus, for this particular KBFE prototype, the exercise is expected to last around nine to ten months.
- landmark for KBFE prototype: the aim of the initial development is to demonstrate the functionality of the envisaged final version and to indicate, if not actually implement, several possible presentation forms.
- identification of end user sites, including banks and other commercial organisations, some of which to participate in the evaluation/assessment plan (see below).
- the evaluation/assessment plan itself which requires that a stage-by-stage monitoring of the KBFE as it is developed. The plan is devised in consultation with evaluation specialists operating within the appropriate work package within the FOCUS package.

6 Conclusions

It is premature to draw firm conclusions at this stage of the FOCUS project; though several operational KBFE prototypes exists and are available for demonstration, and indeed for evaluation, there is as yet, insufficient evidence to make claims of substantial success. However, participants in, and observers of, the FOCUS project are prepared to attest to their growing belief that it is indeed possible to develop worthwhile generic technology for front-end construction, and that the scope of applicability of that technology is perhaps even greater than originally envisaged. The KBFE exercise described in some detail in section 5 is typical of the progress now being made within the project, as underlying tools and technologies stabilise. That particular exercise is still in the construction phase but others have undergoing intensive evaluation. Preliminary reports from those evaluation activities serve to bolster the confidence expressed below.

7 Acknowledgements

The funding support of the European Commission is carrying out the work of the FOCUS project is gratefully acknowledged. The author also wishes to acknowledge the contributions of several FOCUS participants in the use of excerpts from FOCUS documentation in this paper. In particular, he is grateful to Professor Albert Prat and his colleagues, of the Universitat de Politecnica de Catalunya, Barcelona, for permission to use an abridged form of the time series KBFE description.

Transcript editing, a simple user interface

Niklas Holsti
Department of Computer Science
University of Helsinki

Abstract

The simplest user interface to write is one where the program reads the input data, computes, and then prints the output data. Such a program is often awkward to use interactively. We demonstrate a tool, the transcript editor, and programming methods that make it radically easier to interact with read-compute-print programs. An algebraic calculator program is shown as an example; a version of Matlab has also been implemented.

1 Transcript editing

A transcript is simply the combination of the input and output data of an interactive session, interleaved in the natural way. For example, the input might be Matlab commands and the output Matlab responses. A transcript editor is similar to a text editor. It lets the user change the input part of a transcript, for example change an earlier command. The editor then cooperates with the application program to update the output part of the transcript so as to make it consistent with the new input. This incremental input-output connection has the obvious function of correcting errors, but it also manages dynamic, pop-up information such as menus and help.

Although more sophisticated incremental processing methods could be used, the method that best suits conventional programming style is a combination of reversal (undoing) and reprocessing (redoing). When the user changes some input, the editor asks the application program to reverse, or undo, the processing of the changed input. The application program returns to a state that was in effect at some point before the changed input. Starting from that point the editor resubmits the input to the application program. The editor puts the new output into the transcript, discards the invalid part of the old output, and updates the screen to show the new state of the transcript.

For large computations, the undo/redo method can be made efficient by structuring the transcript as a hypertext, a directed acyclic graph of text blocks.

Archer et al [1] use *script* editing as a formal description of undo/redo systems, where the term script means just the input data. However, they suggest that it would be "too powerful and confusing for general use". Our approach [2,3,4] uses *transcript* (input-output) editing as a general interaction tool. This tool supports the usual dialogue forms and at the same time provides implicit and global undo/redo functions. We feel that it is suitable for most interactive situations where inputs are textual.

2 Application programming

Writing application programs to run under a transcript editor resembles batch programming more than interactive programming. The form and order of the input data can be fixed, since the editor provides random access to any part of the input at any time. If the program finds an error in the input, it need only emit an error message and stop (wait for a reversal request) - no other error-correction or input-editing dialogues are needed. Such error messages can easily be made into a context-sensitive pop-up help facility.

Interactive programs often provide special user commands to output selected results, whereas under a transcript editor, output can be produced in natural batches and analyzed with the editor's scrolling and string searching functions.

When the script is changed, the state to which the application program reverses does not have to be immediately before the point of change - any earlier position will do. Thus, the programmer can select how coarse or fine the reversal and reprocessing grains will be. For example, it is trivial to reverse the whole computation and just start over, whatever part of the input the user changes. Response speed is increased if the program reverses to, and recomputes from, some intermediate checkpoint state, but the grain size has no other effect on the user - the amount of typing and the mental effort stay the same.

3 Implementations

We think of the application program as parsing the input data, whether the inputs are commands, menu selections, or replies to questions posed by the program. Typically, the program will perform its computations during parsing, for example using a stack to evaluate an expression entered by the user. When the user changes some input, the parsing and computation must be reversed to an earlier state.

Using the freedom to choose the reversal grains, as noted in the preceding section, we have found simple methods for reversing parsing and parser-driven computation [4]. A bottom-up parser can be made reversible either with a grammar transformation or with a modified parsing algorithm; in both methods the parse stack is used as a history list. A top-down, recursive descent parser can use the procedure call stack as a history list. Recursive descent parsing is easy to reverse in a language with exception handling, for example Ada. In other languages, exceptions can be simulated with moderate effort. The reversible parsing methods extend to the reversal of any computation on stacks. This covers most of the operations of a typical interpreter/compiler: symbol tables, generated code, and intermediate result stacks.

The calculator program we demonstrate is written in Turbo Pascal as a recursive descent interpreter, using simulated exceptions. The Matlab implementation [3,4] used an LALR(1) bottom-up parser produced by the HLP parser generator [5].

We have implemented transcript editors for various systems and languages. The one we demonstrate is written in Turbo Pascal. It is demonstrated on a Macintosh, but it is fairly portable. A transcript editor in C++ on PC's is nearing completion. On VAX/VMS we have an extension of the DEC EVE editor; this transcript editor runs the application programs in subprocesses, so the applications can be written in any programming language. In contrast, the Macintosh and PC editors must be linked to the application program, which will usually by written in the same language as the editor (Pascal or C++).

A transcript editor can also be used with existing non-reversible programs, although the global consistency of the transcript is not ensured. Our VAX/VMS editor has been used extensively with the Macsyma program, for example.

4 Conclusion

Transcript editing is a simple route to comfortable interaction with conventional read-write statements. Documentation and reproducibility are basic requirements of scientific computation. Transcript editing supports them by storing all user inputs in their final form and ensuring that output is also up to date, rather like a 'make' facility.

Whatever tortuous path of interactive trial and error is followed, the edited transcript remains a clear and consistent record of the computation.

Acknowledgements. My warm thanks to Juha Vihavainen, P¡ae¿aivi Kuuppelom¡ae¿aki and their students for the PC/C++ implementation. My work was supported by the Finnish Ministry of Education through its doctoral program in computer science.

References

[1] J. Archer, Jr., R. Conway, and F. Schneider, "User recovery and reversal in interactive systems," *ACM Transactions on Programming Languages and Systems* 6, 1 (January 1984), 1-19.

[2] N. Holsti, "Incrementality in problem solving with computers," in *Problem Solving Environments for Scientific Computing* (Proceedings of the IFIP TC2/WG2.5 Working Conference, Sophia Antipolis, France, 17-21 June 1985, edited by B. Ford and F. Chatelin), Elsevier 1987, 317-324.

[3] N. Holsti, "A session editor with incremental execution functions," *Software - Practice and Experience* 19, 4 (April 1989), 329-350.

[4] N. Holsti, "Script editing for recovery and reversal in textual user interfaces" (PhD Thesis), Report A-1989-5, Department of Computer Science, University of Helsinki.

[5] K. Koskimies et al, "The design of a language processor generator," *Software - Practice and Experience* 18, 2 (February 1988), 107-135.

Investigations Into the Design of Very High-Level Interactive Interfaces for an Automated Numerical Analyst

by

Fred Grossman [a], Melvin Klerer [b] and Robert J. Klerer [c]
[a]School of Computer Science & Information Systems
Pace University, 1 Martine Avenue
White Plains, NY 10606
USA

[b]Department of Computer Science
Polytechnic University, 333 Jay Street
Brooklyn, NY 11201
USA

[c]Department of Computer Science
Iona College, 715 North Avenue
New Rochelle, NY 10801
USA

Introduction

We will consider various design issues involved in the eventual development of an AUTOMATED NUMERICAL ANALYST. As a platform for this development we will use the AUTOMATED PROGRAMMER, a robust system that automates a great deal of the mundane details of conventional scientific, engineering, and mathematical applications programming (1, 2). We will extend the framework of strategies that we have previously developed for the AUTOMATED PROGRAMMER to the problem-specific areas encompassed by an AUTOMATED NUMERICAL ANALYST.

The AUTOMATED PROGRAMMER is a language, or more precisely, a language system, in which programming can be avoided completely in some cases, and considerably moderated in other cases. This is accomplished by the conceptually simple device of accepting programs (executable modules) which are entered using conventional (pre-computer) textbook two-dimensional mathematical notation and control structures, characterized by a flexible syntax, which mimic technical English. If the problem solver has a well-formulated statement of a computational algorithm expressed in mathematical notation and technical English, such as can be found in numerical analysis texts, then the AUTOMATED PROGRAMMER system minimizes the tra~~n~~sformation to an executable program. It supports desirable attributes for enhancing documentation, error avoidance maintainability, and verifiability. Since it also decreases programming time (3), this approach tends to substantially ameliorate the programming requirements for scientific problem solving.

Figure 1

For $n = 3$ by 3 to 9 and $\alpha = 2$ to $n^2 - 2$ and $\beta = 2(2)n$ read r, g, μ, if $\frac{g}{n} < p$ then

$$t_{\alpha\beta} = \int_{2}^{n} \int_{3}^{\sqrt{\frac{y}{n}}} \frac{\frac{e^{-\mu x}}{y+n} \sum_{i=1}^{n} \sum_{j=1}^{n} \left\{ \cos\ y + \frac{\sin^{i+j} x}{(i+j)x} \right\}}{\frac{\mathrm{LOG}_r\ x\ +\ \mathrm{TAN}^{-1}\ \frac{x^n}{y^{n-1}}}{\sqrt{\frac{\alpha}{\beta+\frac{x}{y}} + \frac{\alpha+\frac{y}{\beta}}{x}}} + \prod_{k=1}^{n^2-\alpha} \left[\frac{\beta^k}{k} + \sqrt{(x+y)^{-k}} \right]} dx\ \ dy$$

and print $\alpha, \beta, n, t_{\alpha\beta}$ else $y = e^{\alpha}\beta^3 - \mathrm{SIN}^2\beta\ \mathrm{COS}\ \beta + 1 - \frac{\mathrm{TAN}^{-1}\beta}{1-|\beta|}$, print β, y. end.

This point of view is illustrated in Figure 1, an example of a definite integral. It is not only a problem solution specification expressed in conventional mathematical notation, it is also an executable document acceptable as input by the AUTOMATED PROGRAMMER. In contrast, the programming of the integrand using a conventional programming language, e.g., FORTRAN, requires considerable attention to syntactical detail and is a non-trivial programming effort.

The AUTOMATED PROGRAMMER has been designed to minimize the linguistic difference between the solution specification, expressed in a notation and syntax "natural" to the application area, and the executable program module which is computationally equivalent to the specification. The resulting program can be considered to be "self-documenting" in the sense that the program representation is largely intelligible to other practitioners in the application field who may have either no or limited knowledge of the few specific programming artifices that may be present in the document.

Automating Numerical Analysis and Computation

Currently, when referencing the numerical analysis literature or seeking to use the extensive mathematical software libraries which are available, computer users are confronted with massive documentation of the elaborate criteria necessary to make an appropriate choice of an algorithm or software procedure. Available software routines usually require the user to conform to complex and extensive parameter passing protocols and to code, in some programming language, a great deal of ancillary information in order to properly invoke the selected routine. With the AUTOMATED PROGRAMMER we have already demonstrated that this latter burden is unnecessary for scientific application programming.

Figure 2

Gauss-Legendre Quadrature - Fifteen point formula with unequal spacing

function $f(x) = \frac{x \sin 2x}{\sqrt{1-(\frac{x}{2\pi})^2}}$.

$z_2 = 0.20119409$, $z_3 = 0.39415135$, $z_4 = 0.57097217$, $z_5 = 0.72441773$, $z_6 = 0.84820658$, $z_7 = 0.93727339$, $z_8 = 0.98799252$.

$w_1 = 0.20257824$, $w_2 = 0.19843149$, $w_3 = 0.18616100$, $w_4 = 0.16626921$, $w_5 = 0.13957068$, $w_6 = 0.10715922$, $w_7 = 0.07036605, w_8 = 0.03075324$.

$a = 0$, $b = \pi$, $c = \frac{b-a}{2}$, $d = \frac{b+a}{2}$.

integral $= c\left[w_1\ f(d) + \sum_{j=2}^{8} \left\{w_j(f(d+cz_j) + f(d-cz_j))\right\}\right]$. print integral.

Consider the task of finding the value of a definite integral of a well behaved function by one of the several available numerical integration methods. One could peruse a textbook to find a suitable method and then construct a program. For example, Figure 2 is an AUTOMATED PROGRAMMER executable specification (program) to integrate an integrand represented by the function f. Presumably, $f(x)$ would normally be defined in a main program which will call the integration subprogram. An alternate approach would be to use a mathematical library routine which is available from public domain or commercial vendors. Typically, this might involve a subroutine call as illustrated in Figure 3.

Figure 3

Integration using NAG Library routine D01AJF

origin 1.

External subroutine D01AJF (function,,,,,,, 1, int, int 1, int, int).

function $f(x) = \frac{x \sin 2x}{\sqrt{1-(\frac{x}{3\pi})^2}}$. $a = 0$, $b = \pi$, $\epsilon abs = 0$, $\epsilon rel = 10^{-4}$.

result = 0, abserr = 9999, ifail = 0. lw = 800, liw = $\frac{\text{lw}}{8} + 2$. dimension w_{800}, iw_{102}.

call D01AJF (f, a, b, ϵabs, ϵrel, result, abserr, w, lw, iw, liw, ifail).

print "Result = ", result, "Absolute error = ", abserr.

While using a library module eliminates the need for the user to program a standard routine, it does require a detailed rigorous protocol to specify programming attributes for each parameter. This usually results in a substantial amount of "boiler plate" preceding the actual invocation of the library subprogram. Nevertheless, this approach does not relieve the user from programming, in detail, the function (integrand), e.g., the first actual parameter of the called routine in Figure 3.

Very recently, there has been interest in researching and developing systems that would ease the user's task of selecting and monitoring appropriate numerical solutions to problems typically covered by conventional numerical computation methodology. Several investigators have approached segments of this topic from the viewpoint of expert system development (5, 6). Others have emphasized the development of knowledge bases that could be used with currently available libraries of numerical procedures (7). Workers in this field, however, have adopted varying emphases toward researching and developing an appropriate man-machine interface for communication and interaction between user and the computer software system.

The High-Level Interface

We believe that the development of an appropriate user interface is crucial for ensuring the pragmatic feasibility of any system whose goal is to automate a substantial part (or all) of the decision processes involved in scientific computation. Regardless of the degree of expertise and the extent of the knowledge base of any of the proposed systems, their practical utility would be severely limited if their user interface were not sufficiently robust to encompass a wide spectrum of users ranging from the novice engineer, untutored in the subtle aspects of the craft of numerical analysis, to the highly sophisticated and expert applied mathematician who comes prepared to his task with an armada of knowledge and experience.

Since the domain of discussion is applied mathematics, we suggest that an acceptable user interface must have the capability of understanding, i.e., recognizing, mathematical input expressed in conventional (textbook) two-dimensional mathematical notation and problem descriptions phrased in technical English distinguished by broad syntactic and lexical flexibility. The design of such a capability raises various questions as to how to resolve the ambiguous formulations that the user may generate given such flexibility for input. Strategies must be developed that take into account the global context dependencies of the presented problem, the accumulated "knowledge" resident in the machine software system, and the development of useful interfaces to deliver to the user not only the results of the analysis achieved by the machine system but also appropriate questions, cautions, and warnings that might limit the validity of the results obtained. That is, the design of the interaction strategy between machine and user should be based not only on purely technical considerations, but also on a psychological model of the user that is subject to experimental verification and modification if the experimental results should so indicate.

We believe that we have solved much of the user-interface problem for a system whose goal is the automation of scientific, engineering, mathematical application programming with the AUTOMATED PROGRAMMER (2). In developing the AUTOMATED PROGRAMMER we have successfully dealt with the problems of input ambiguity, context-dependent analysis, feedback of system interpretation to the user, and the validity of the results produced by a numerical computation – all within the context of application programming given fairly well specified computational algorithms. The design of an interactive interface to an AUTOMATED NUMERICAL ANALYST poses more difficult problems in the extensions of these concepts. In all instances, the problem-solver should be able to create, maintain, reuse and verify his problem specifications in language and notation natural to his mathematical-scientific problem solving domain, irrespective of the target execution environment.

The AUTOMATED PROGRAMMER already has the input capability of displaying ordinary and partial differential equations in their conventional two-dimensional form, and we believe that extending the recognition algorithms and heuristics presently employed by the AUTOMATED PROGRAMMER so that these equations may be recognized will be straight forward. Figure 4 gives various examples of how such input might be presented to an AUTOMATED NUMERICAL ANALYST.

Figure 4

Solve $\frac{dy}{dx} = y - x$, initial $y = 2$ at $x = 0$ for $x = 0.1, 0.2, ..., 2$.

Solve $\frac{dy}{dx} + y^2 = 0$, initial $y(0) = 1$ for $x = .1, .2, .3, .4, .5$.

Solve $\frac{d^2y}{dx^2} - 2y^3 = 0$, initial $y(0) = 1, y'(0) = -1$ for $x = .1, .2$.

Solve $\frac{\delta^2 u}{\delta x^2} + \frac{\delta^2 u}{\delta y^2} = (x^2 + y^2)e^{-xy}u^2$ with $0 < x < 1$ and $0 < y < 1$

and boundary conditions $(u = 1$ for $x = 0, y = 0)$, $(u = e^y$ for $x = 1)$, $(u = e^x$ for $y = 1)$.

Concomitant to this would be the consideration of strategies for machine understanding of the specific problem when posed by the user in technical English. We would anticipate that our approach here would consist of the design of a set of strategies specific to each problem area. This involves studying how, for specific problem areas, problems are actually posed in textbooks and the professional literature. Thus our underlying assumption is that even the user who can be regarded as naive in the intricacies of the craft of numerical analysis will none-the-less have sufficient acquaintance with the way problems are posed, i.e., he will have some degree of mathematical literacy. Thus this places some bounds on the proposed flexibility to be expected in understanding what the user poses as a problem. However the problem of automated understanding, while bounded, is still a substantial research issue.

It is necessary to develop further insight into:

(a) the limits to the flexibility and extensibility of two-dimensional input-output via computer

(b) the limits to syntactic and lexical flexibility of user problem input phrased in technical English

(c) the extent to which a knowledge base is necessary for a robust system for specific problem areas in scientific and mathematical computation.

For example, consider the partial differential equation shown at the bottom of Figure 4. An example of an ELLPACK program for this problem is given in (16). An automated understanding of this problem would entail (1) the recognition that the function to be found is $u(x, y)$, (2) the translation of the two-dimensional mathematical text into an equivalent linear formulation (e.g., Fortran), (3) the generation of the target code in ELLPACK format. Thus, an AUTOMATED NUMERICAL ANALYST might generate the following ELLPACK interface target code:

Equation

```
UXX + UYY = (X ** 2 + Y ** 2) * EXP(-X * Y) * U(X,Y) ** 2
```

Boundary

```
U = 1.0         ON X = 0.0
                ON Y = 0.0
U = EXP(Y)      ON X = 1.0
U = EXP(X)      ON Y = 1.0
```

Conclusion

We propose design considerations for a high-level user interface to a knowledge framework which would allow the easy incorporation, in machine understandable terms, of the wisdom already available in numerical analysis literature and practice relevant to the selection of an appropriate solution, given that the characteristics of a problem are understood. Such an interface module should be suitable for the intelligent selection of routines from available libraries of procedures (e.g., NAG, IMSL, etc.) We put emphasis on the terms easy and intelligent. That is, although our approach is pragmatic and problem specific, our goal is to create a framework that is flexible and expandable to new knowledge. For example, envision a system in which the user will be able to browse through the available library routines (external or intrinsic) and get information about the mathematical assumptions and computational characteristics of each function. Upon selecting the appropriate library function, the user will be guided through the specification of necessary or overridable default parameters. All interaction with the system will be in normal mathematical notation and technical English. There will be no need for a user to know the number or order of any required parameters. When the routine is executing and a possible mathematical exception or anomaly arises, the user will be guided through alternative choices of algorithms, procedures or parameters in order to continue the computation.

Our strong advantage in pursuing this proposed research is that we already have a robust "engine" (the AUTOMATED PROGRAMMER) that is expert enough to recognize and understand conventional two-dimensional mathematical notation, as used in the numerical analytical literature, and can accept a much broader range of syntactic and lexical structures than that permitted by conventional high-level programming languages.

We believe that we can extend these capabilities to the more difficult problem of providing an appropriate interactive man-machine interface for a system whose goal is the automation of numerical computation. We believe that we can extend many of the user-oriented methods that we have developed for the implementation of the AUTOMATED PROGRAMMER to deal with the problems of interaction and feedback to the user, selection of appropriate algorithms expressed as software procedures, elimination of the necessity for the user to create elaborate protocols for the selected procedures, and the validation of computed results.

We anticipate that our efforts toward designing a module appropriate for analyzing the problem framed by the user, the design of a decision process, and the design of an appropriate knowledge-base would be very heavily influenced by the knowledge that already exists in the numerical analytic literature. Our contribution would be to determine how this knowledge can be transformed into a form that not only can be codified, but can also satisfy the user-oriented and interactive attributes necessary for the eventual realization of an AUTOMATED NUMERICAL ANALYST.

References

[1] F. Grossman, R.J. Klerer, and M. Klerer, "A Language for High-Level Programming of Mathematical Applications", Proceedings of the IEEE Computer Society 1988 International Conference on Computer Languages, October 9-13, 1988, Miami, FL, USA, pp. 31-40.

[2] M. Klerer, F. Grossman, and R.J. Klerer, "The Automated Programmer System: Language Design Issues for Scientific-Mathematical-Engineering Application Programming" in The Role of Language in Problem Solving (2), Boudreaux, J.C., Hamill, B.W., and Jernigan, R., Editors, North-Holland Elsevier, 1987.

[3] M. Klerer, "Experimental Study of a Two-Dimensional Language vs Fortran for First-Course Programmers", International Journal Man-Machine Studies, 20, 445-467, 1984.

[4] M. Klerer, Design of Very High-Level Computer Languages - A User-Oriented Approach. McGraw-Hill, 1991.

[5] J.R. Rice, et al, The Second International Conference On Expert Systems For Numerical Computing, Purdue University, Computer Science Dept. Tech. Report CSD-TR-963, March, 1990.

[6] J.R. Rice, et al, IMACS Conference on Expert Systems for Numerical Computing, Purdue University, Computer Science Dept. Tech. Report CSD-TR-827, November, 1988.

[7] J. Chelsom, D. Cornali, and I. Reid, "Numerical and Statistical Knowledge-Based Front-Ends", preprint April 19, 1990 (abstract in ref. [5].

[8] C.E. Houstis, et al, "ATHENA: A Knowledge Base System for ELLPACK", preprint December 28, 1989 (abstract in ref. [5]).

[9] W.R. Dyksen and C.R. Gritter, "Scientific Computing and the Algorithm Selection Problem", preprint (abstract in ref. [5]).

[10] B. Ford, S.J. Hague, and R.M. Iles, "Numerical Knowledge-Based Systems" (in ref. [6]).

[11] D. Kahaner, "Experiences With an Expert System for ODEs" (in ref. [6]).

[12] F. Rechenmann and B. Rousseau, "A Development Shell for Knowledge-Based Systems in Scientific Computing", preprint (abstract in ref. [5]).

[13] A.D. Kowalski, M.F. Russo, and R.L. Peskin, "An Effective Knowledge Representation Scheme for Automatic Numerical Program Generation", preprint (abstract in ref. [5]).

[14] P.W. Gaffney, et al, "NEXUS: Towards a Problem Solving Environment (PSE) for Scientific Computing", ACM SIGNUM, vol. 21, no. 3, July 1986, p. 13 ff.

[15] A. Kaname, M. Chiba, and A. Mochida, "An Approach Toward Integrated Algorithm Information System", Information Systems, vol. 9, no. 3/4, 1984, p. 197 ff.

[16] R.F. Boisvert, "Languages and Software Parts For Elliptic Boundary Value Problems" in The Role of Language in Problem Solving 2, Edited by J.C. Boudreaux, B.W. Hamill, and R.C. Jernigan, North Holland-Elsevier Science Publishers, Amsterdam and New York, 1987.

A Visual Language for Program and Reasoning Flow

D. Y. Y. Yun and S. Chen
Electrical and Computer Engineering
492 Holmes Hall, 2540 Dole Street
University of Hawaii at Manoa
Honolulu, Hawaii 96822

ABSTRACT

In practical programming systems the ideal of logic programming has not been achieved because logic and control cannot be completely separated in practice. In Prolog, the most successful and the most widespread logic programming language, programmers must still be concerned with the procedural control flow of programs for reasonable efficiency and desired side effects. A fair amount of high level control constructs are needed for expressing the procedural semantics of a logic program. The goal of this work is to establish a representation, a visual language, and some system support mechanisms for assisting programmers in handling the difficulties arising from these unavoidable control issues in logic programming.

In this paper, a visual programming model for dealing with the control flow of logic programs is presented, together with a representation model called VCF (Visual Control Flow). Existing Prolog programs may be represented in a VCF diagram which is then transformed to make run time control flow explicit. With these transformed diagrams, further modifications may be applied to improve the efficiency of execution as well as the understandability of the program. Used as a graphical programming language in the design phase, VCF diagrams provide programmers with effective means to express the logical concepts of programs directly. Finally, VCF diagrams may be used as inputs to a compiler for more efficient code generation and as a visually interactive debugging tool.

1. INTRODUCTION

Rapid prototyping of intelligent programs is essential to achieve better acquisition, management and utilization of problem solving expertise. A step toward this goal is accomplished by visualization of reasoning flow in a logic programming environment. A graphic language simplifying the visualization of reasoning flow in logic programs is presented. In the proposed Visual Control Flow (VCF) representation, visual diagrams present to users the flow of control in a given logic program. This VCF allows a user to ensure the proper utilization of knowledge and even achieve unexpected efficiency from automatic reduction heuristics of the support system. VCF is also extended into a hierarchical reasoning flow (HRF) model that is capable of representing an entire knowledge base of logic programs, each represented hy a VCF diagram. The reasoning flow information of a logic program at a specific level of abstraction is completely reflected by the diagram. This diagram contains all the possible effects of truth value assignment of its program units.

Applying the HRF model to a knowledge base allows easier expression and understanding of logic programs that manage and utilize all data and knowledge. These programs are expressed in terms of rules and facts dealing with the "features" of the underlying application domain. These features are, in turn, represented in the form of symbolic names, properties, relations, and concepts, each of which could be (recursively) a logic program unit as well.

Logic programming is an established tool for symbolic processing, reasoning, and problem solving in Artificial Intelligence (AI). Adopting the first order predicate logic [Warren82], this approach is entirely user-oriented [Kowalski74]. This field of using logic for programming has seen a tremendous growth in the past decade both in depth and in scope [Shapiro87]. The increasing popularity comes primarily from the knowledge expressive power in the declarative well-formed-formula (WFF) representation and from the automatic resolution-unification [Robinson65] deductive reasoning mechanism.

The uniform knowledge representation of first-order predicate logic allows programmers to concentrate primarily on the development of the knowledge base (KB) for a chosen application domain. Separating the explicit problem solving knowledge from the problem solving mechanism its most significant aspect. Conventional programming languages normally require users to plan a sequence of imperative actions to handle match-and-act executions in different levels. Relieving the users (programmers) from the burden of expressing imperative details, while allowing him to focus on the expression of knowledge and strategy declaratively, is the fundamental difference, hence advantage, of logic programming. The solid logic foundation of its reasoning mechanism is completely automatic, freeing the user from the concerns of details in execution controls.

The original logic programming paradigm intentionally ignores all the explicit control constructs. Conjunction of ordered goals and Disjunction of ordered clauses are two underlying constructs. As a practical AI language, Prolog had been compromised [Clocksin87] to include a few primitive control structures (such as cut, fail, repeat, abort, and halt) in order to gain more efficiency and to achieve desirable side effects. As a result, the flow of control in reasoning through a knowledge base can be difficult to follow and hard to manage, especially for inexperienced programmers. Such difficulties can be illustrated by the following logic program for merging two already sorted lists:

```
merge([], L, L) :- !.
merge(L, [], L).
merge([A|B], [C|D], [A|E]) :- A =< C, !, merge(B, [C|D], E).
merge([A|B], [C|D], [C|E]) :- merge([A|B], D, E).
```

It is not easy to predict the actual flow of reasoning from one statement to another during program design or during coding phase. It has been shown that some assistence in visualizing the reasoning flow of the logic program can significantly enhance the understandability of the code, the correctness of a developing program, and also the productivity of programmers. Iconic and visual representations, as opposed to linear textual structures, of programs have been proven valuable to human understanding as well as human associative and creative thinking [Chang87].

2. A VISUAL FLOW MODEL

Normally, each logic program is abstractly identified by a name and several formal parameters in the form of a predicate function term as a defined concept which describes different possible methods to achieve a class of goals (queries). A goal represents some instance(s) of the predicate term. Rules and/or facts are enumerated as a collection of the clauses describing the methods. Basically, a rule is a clause describing a method in which several subgoals need to be achieved first in order to achieve the goal. A clause without subgoals is called a fact describing some goal(s) which can be achieved directly. Based on these structural requirements, it is quite evident that a Visual Control Flow (VCF) diagram would be helpful to logic programmers. For example, the merge program above can be directly visualized as a VCF diagram in top part of Figure 1.

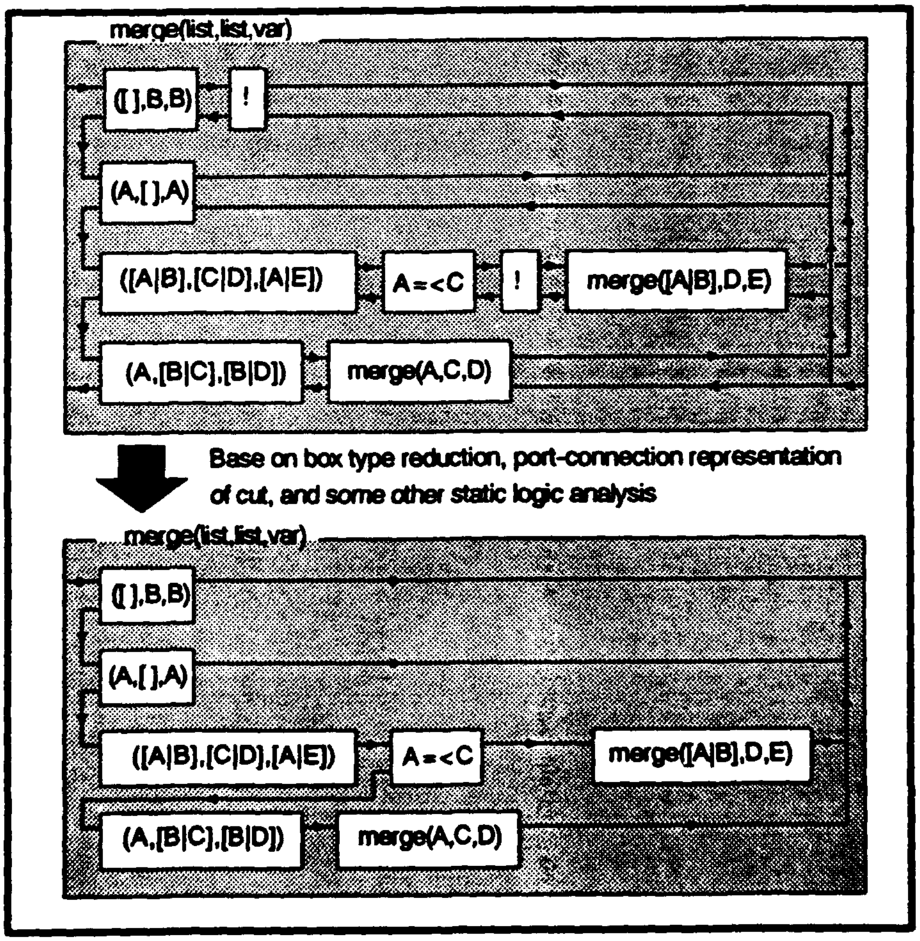


Figure 1: VCF Diagram for Merge and its Reductions

Using these VCF diagrams as building blocks, a Hierarchical Reasoning Flow (HRF) model for expressing and visualizing an entire knowledge base of logic programs can be constructed. In the HRF model, a goal (or a program) will be represented as a labeled box with at most four control ports. To visualize the internal details of this logic program, the immediate next lower level of abstraction will be seen as an expanded box, also with at *most four ports*. The expanded box may include several other boxes recursively which are put together into a network by a set of port-connections. The network will reflect the reasoning flow among goals and subgoals at that level. The nested hierarchy of all goals forms a directed graph, where each node (a concept on its own) is actually a network consisting of lower level concepts. Any higher level box can be opened up to reveal details in the next immediate lower level, which is a port-connection network of internal boxes reflecting the reasoning flow of all the rules/facts. Figure 2 is a typical HRF diagram, showing the clauses and control flow to solve the (familiar) N-Queens problem.

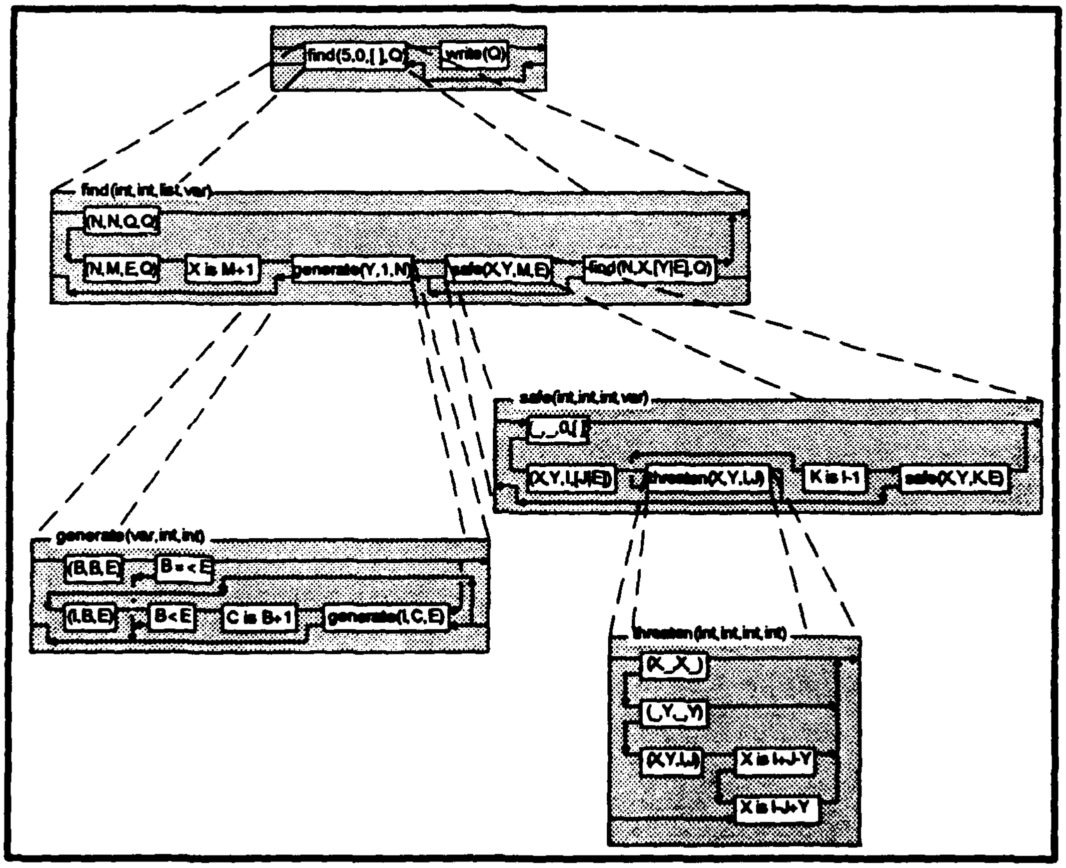


Figure 2: HRF Model for the N-Queens Problem

The HRF model allows users to understand an existing knowledge base by visualizing its actual reasoning flow in a hierarchical manner. Experience with this reasoning flow visualization has shown that it can facilitate the process of update or modification of the knowledge base as well. Further extensions of it evolve the HRF system into a graphical programming language by which knowledge of utilizing and managing data/knowledge can be expressed and communicated among people via the visualization of programmed logic in moving action (flow of reasoning).

The primitive idea of the 4-port boxes was originally proposed to help understand the goal querying in Prolog [Byrd80]. The primiuve 4-port box model had been also used in the debugger Coda [Plummer88]. In this HRF model, each goal, or indeed each logic program unit, is always presented as a box with four ports; ***call, exit, fail,*** and ***redo***. These four ports represent two types of entries (***call, redo***) and two types of exits (***exit, fail***) indicating the reasoning flow for the predicate in forward (***call, exit***) or backward (***redo, fail***) execution. Similar diagrams have been adopted to trace the reasoning of a logic program at debugging time [Plummer88, Walker87].

3. TRANSFORMATIONS AND REDUCTIONS

In this HRF model, the *full* 4-port box is used only to represent the most general goals or those goals whose properties have not yet been explored. In other words, not all predicates (goals) make use of all 4 ports. Considering the necessity of ports in the 16 possible combinations of the 4-port boxes, only six types of boxes are meaningful (Figure 3). Based on these six types of 4-port boxes, some simplifications can already be made. As an example, if it is known that a predicate has no way to fail, then ignoring the fail port as well as its corresponding fail connection makes the representation clearer and more understandable. Some possible reductions among these primitive 4-port boxes are shown in Figure 3.

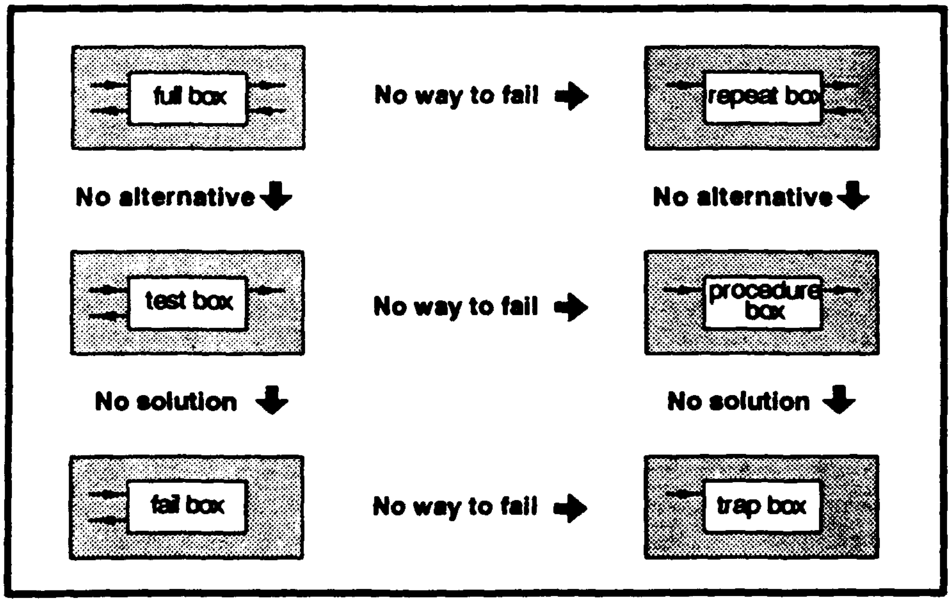


Figure 3: Six Meaningful 4-port Boxes and their Possible Reductions

In addition, some of the more controversial non-declarative control constructs, such as cut, fail, and repeat are represented in their equivalent port connection patterns to reflect the actual side effects. Figure 4 shows the substitutions of the construct *cut (!)* and the predicate *not (\+)*.

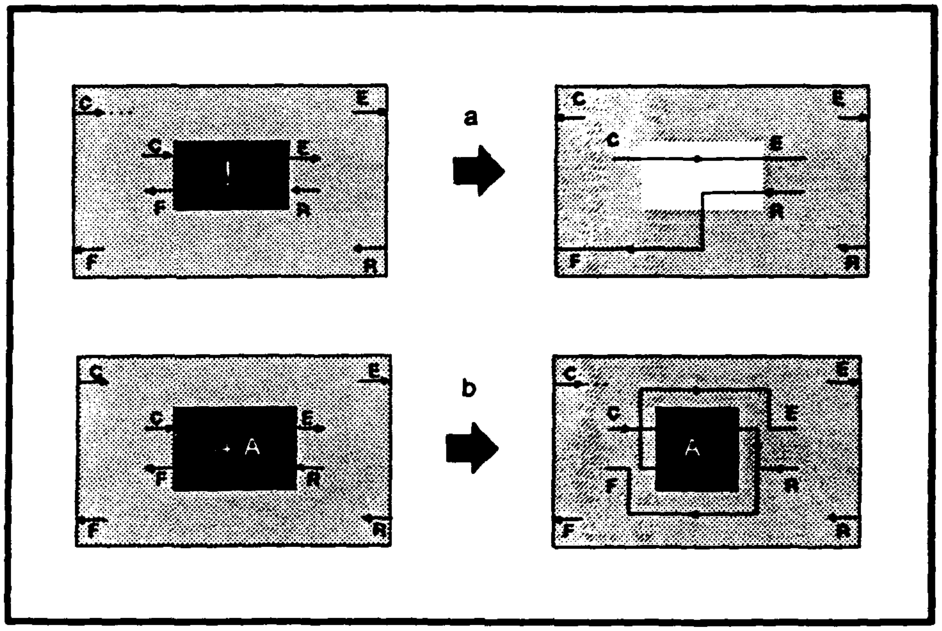


Figure 4. Transformation of Control Constructs

From standard Prolog point of view, using the full box to uniformly represent each individual logic unit is straightforward and effectively describing all possible behaviors of the predicate. It is even more desirable to evolve this naive representation automatically and/or manually so that final reasoning flow diagram can more clearly reflect all the effective choice points and more efficiently avoid entering all the unnecessary internal input ports (call or exit). Basically, such a reduction/simplification process involves automatic transformations derived directly from the box type reductions (in Figure 3), the substitution of non-declarative control constructs (in Figure 4), and some analysis of inter-relationships among clauses (such as mutually exclusive heads) or among subgoals in a single rule (such as certain data dependency). The bottom part of Figure 1 shows the result of applying these reductions the the Merge program at the top.

Subsequent transformations can further enhance the HRF automatically, e.g. skipping unnecessary transfers of control into some 'black' boxes and showing choice points more effectively through the simplified, clearer diagram. More experienced users could also manually modify the HRF diagram to better reveal the programmer's intent through this visualization as well. When some of these optimizing transformations are applied to the HRF for the n-queens problem of Figure 2, improvements can be easily seen in Figure 5.

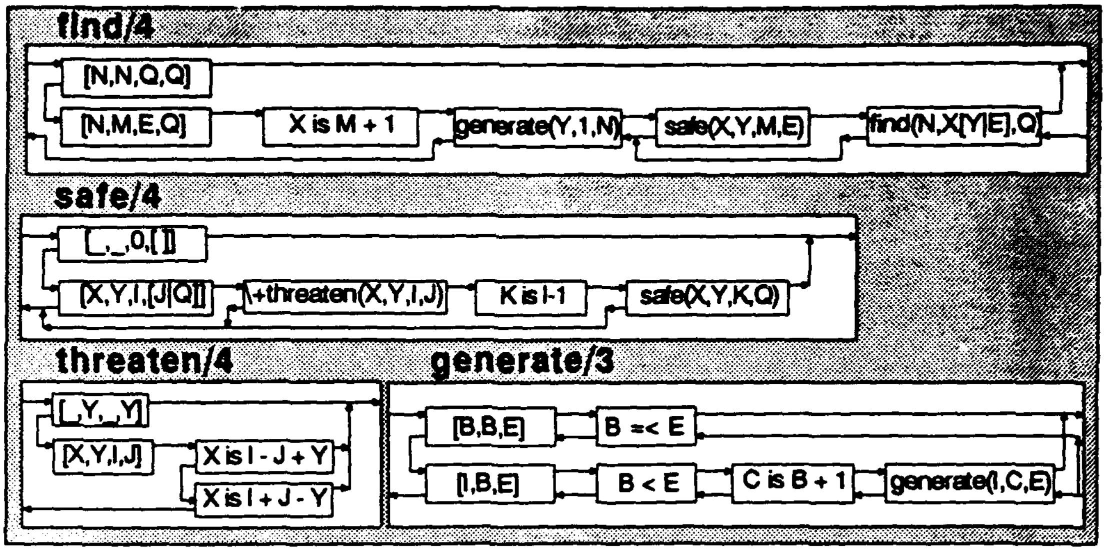


Figure 5: Optimized HRF for the N-Queens Program

4. A PROGRAMMING ENVIRONMENT

In using Prolog as a language for prototyping intelligent programs, it is often unavoidable that some undesirable, inefficient logic constructs would be used. This difficulty has, indeed, been considered hindrances to elevate Prolog to an even more successful AI language. The HRF model, with its automatic and manual reduction capacity that simplify and visualize control flow, can be used as a debugging and learning tool. Such assistances could be especially helpful to any programmer with difficulties adjusting to the peculiar reasoning flow of logic programs. HRF diagram has been used successfully as a visual language to express and convey user's logic (control) intents. Based on the visual HRF diagram, a prototype of enhanced logic programming environment is depicted in Figure 6, including six subsystems, namely Logic-Program Converter, Interactive Diagram-Editor, Automatic Diagram-Improver, Diagram-Quality Analyzer, Dynamic Intelligent-Debugger, and Executable-Code Generator.

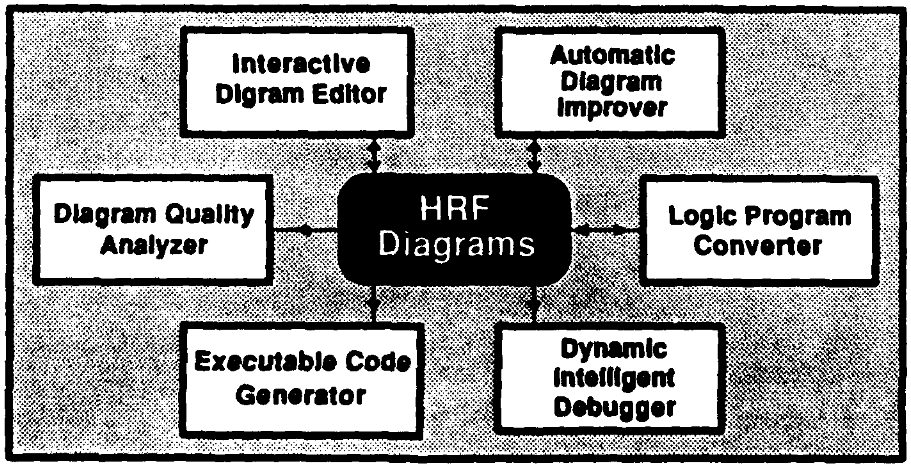

Interactive Digram Editor
Automatic Diagram Improver
Diagram Quality Analyzer
HRF Diagrams
Logic Program Converter
Executable Code Generator
Dynamic Intelligent Debugger

Figure 6: Visual Logic Programming Environment

To begin using this environment, a HRF diagram can be either translated from an existing textual logic program by the Logic-Program Converter or directly constructed from a conceptual design by the Interactive Diagram-Editor. Any naive HRF diagram will be automatically transformed by the Automatic Diagram-Improver to remove any unnecessary boxes, ports, and/or connections, so that users may clearly visualize the effective reasoning flow of the program under development. The Diagram-Quality Analyzer statistically reveals the effectiveness of automatic reductions in terms of number of ports or number of connections removed. Aided by the benefits of visualization and analyses users may use the Interactive Diagram-Editor to manually adjust the HRF diagram, from time to time, to better express design intentions. Should the diagram get too complex to be understood, the Dynamic Intelligent-Debugger can be used to hierarchically trace the effects of the programs at different levels of abstraction. Finally, effective executable programs will be produced by the Executable-Code Generator, which *compiles* code directly from the HRF diagram.

5. OUTLOOK

Details of the HRF model, as well as each of the six support subsystems, must be further developed. The contributions from each subsystem towards the overall success of the logic programming environment must be individually and collectively demonstrated. Finally, the benefit of utilyzing the HRF model must be proven via statistics, analyses, and, most importantly, user satisfaction. The parallelization of the HRF model and the support environment may also be needed to achieve ever increasing performance requirements.

REFERENCES

[Bowen82] D. L. Bowen, L. Byrd, F. C. N. Pereira, L. M. Pereira, and D. H. D. Warren, *DEC system-10 Prolog user's manual*, Department of Artificial Intelligence, University of Edinburgh, Scotland (Nov.

[Bush88] W. R. Bush, G. Cheng, P. C. McGeer, and A. Despain, "Experience with Prolog as Hardware Specification Language", *IEEE Symposium on Logic Programming*, 490-498 (1987).

[Byrd80] L. Byrd, "Understanding the control flow of Prolog programs", *Proceedings of the 1980 Logic Programming Workshop*, S-A Tarnlund (ed.), 127

[Chang85] J. H. Chang and A. Despain, "Semi-Intelligent Backtracking of PROLOG Based on Static Dependency Analysis", *Proceedings of the IEEE symp. on Logic Programming*, 10-21 (1985).

[Chang87] S. K. Chang, "Visual Languages: A Tuturial and Survey", **IEEE Software**, 4 (1) 2-39 (Jan. 1987).

[Chen87] G. Chen and M. H. Williams, "Executing Pascal programs on a Prolog architecture", **Information and Software Technology** 29 (6) 285-290 (July 1987).

[Clocksin87] W. Clocksin, "A Prolog Primer", **BYTE** 147-158 (Aug. 1987).

[Cohen88] J. Cohen, "A view of the origins and development of Prolog", **Communications of ACM** 31(1) 26-36 (Jan. 1988).

[Covington88] M. A. Covington, D. Nute, and A. Vellino, **Prolog Programming in Depth**, Scott, Foresman and Company (1988).

[Eisenstadt88] M. Eisenstadt & Brayshaw, "The Transparent Prolog Machine (TPM)", **Journal of Logic Programming, (5)** 277-342 (1988).

[Feigenb83] E. A. Feigenbaum and P. McCorduck, **The Fifth Generation Artificial Intelligence and Japan's Computer Carllenge to the World**, Addison-Wesley Publishing Co., vol. 275, 1-1 (Jan. 1983).

[Gregory87] S. Gregory, **Parallel Logic Programming in PARLOG, The Language and its Imp ementation**, Addison-Wesley Publishing Co. (1987).

[Kahn84] K. M. Kahn, "A primitive for the control of logic programs", *Proceedings of the International Conference on Logic Programming* (1984).

[Kowalski74] R. A. Kowalski, "Predicate logic as programming language", *Proceedings of IFIP'74*, North Holland Publishing Co., 569-574 (1974).

[Kowalski79a] R. A. Kowalski, **Logic for Problem Solving**, Elsevier North Holland (1979).

[Kowalski79b] R. A. Kowalski, "Algorithm = Logic + Control", **Communications of ACM**, 22 (7) 424-436 (1979).

[Lloyd84] J. W. Lloyd, **Fundations of Logic Programming**, Springer-Verlag (July 1984).

[Naish85] L. Naish, "All solutions predicates in Prolog", *IEEE Symposium on Logic Programming*, 73-77 (1985).

[O'Keefe85] R. A. O'Keefe, "On the treatment of cuts in Prolog source-level tools", *IEEE Symposium on Logic Programming*, 68-72 (1985).

[Plummer88] D. Plummer, "Coda: An Extended Degugger for Prolog", *International Conference & Symposium on Logic Programming*, 496-511 (1988).

[Pountain88] D. Pountain, "Parallelizing Prolog -- Three approaches to running Prolog programs on multiprocessor machines", **BYTE** 13 (12) 387-394 (Nov.

[Robinson65] J. A. Robinson, "A machine-oriented logic based on the resolution principle", **J. ACM** 12 (1) 23-41 (Jan. 1965).

[Shapiro83] E. Shapiro, "A Subset of Concurrent Prolog and its interpreter", TR-003, ICOT (1983)

[Shapiro87] E. Shapiro, **Concurrent Prolog**, Vol. 1, MIT Press, p.2 (1987).

[Takeuchi86] A. Takeuchi and K. Furukawa, "Parallel Logic Programming Languages", *The 3rd International Conference on Logic Programming*, 242-254 (July 1986).

[Ueda86] K. Ueda, "Guarded Horn Clauses", Doctoral Thesis, Information Engineering Course, Faculty of Engineering, Univ. of Tokyo (1986).

[Warren80] D. H. D. Warren, "Logic programming and Compiler writing", **Software Practice Experience** 10 97-125 (Feb. 1980).

[Warren82] D. H. D. Warren, "Higher-order extensions to Prolog: are they needed?", **Machine Intelligence** 1 441-454 (1982).

Automatic Generation of Optimized Numerical Code for Jacobians and Hessians

F.C. Berger V.V. Goldman M.C. van Heerwaarden J.A. van Hulzen

Department of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands.

Abstract

The main features of GENJAC are presented. It is a package for the automatic generation of numerical code for Jacobians and Hessians. It applies GENTRAN, a code generation program, and SCOPE, a source code optimization package. GENJAC, GENTRAN and SCOPE are extensions of the computer algebra system REDUCE.

1. Introduction. The ever increasing computational power offered by new architectures has broadened the scope of scientific computation. Coupled to this computational power has been the ever increasing need for suitable environments to manage the problem-solving process as a whole. A user interface such as SUI[3,22], currently under development at Kent State University, can serve multiple client systems in parallel, including symbolic, numeric, graphics and document formatting systems. This creates the need for user friendly communication facilities between the different systems. Symbolic, or more precisely, computer algebra systems provide interaction with a user, often based on command interpretation. The intention is to allow a user to simulate paper and pencil computations. Although these systems provide powerful mathematical tools in an "exact world", they are not particulary well suited for doing numerical calculations. This is related to internal data handling strategies and command interpretation. It introduces the need to provide an interface between computer algebra facilities and numerical tools, such as libraries. Output, i.e. mathematical expressions or assignment statements, in the form of syntactically correct code for numerical processing is merely a beginning. One of the findings given in a recent SIAM report [2] is, that the separation between symbolic and numeric computation is still (too) large. Bridging this gap is of urgent need, since the present increase in computational power invariantly leads to increase in problem size and complexity. This -in turn- will lead to numerical programs, which are not longer codable by hand. Then desk-top hardware can be used as a front-end facility for code-production. The need for programming environments, including tools for automatic or semi-automatic program generation, is obvious. Computer algebra can provide essential services, assuming the earlier mentioned symbolic-numeric interface is adequate. Ideally, a problem specification, given in combination with a specification of the target machine and the required software tools, and serving as input leads to code -the output- defined in the speficied software tools and executable on the target machine. Or, even better, it leads to the solution, presented in a user required form. The route from specification to solution requires a number of intermediate stages, each of which demands software tools, combinable with or partly depending on a computer algebra system. The target machine may request problem decomposition. The processes -thus defined- may need code vectorization. In addition these processes ought to be formulated in syntactically correct, optimized and reliable target program source code. Such requirements indicate that a symbolic-numeric interface can be seen as a collection of facilities, which together form the (semi-)automatic program generation features.

Hence, such an interface requires much more than merely translating the description of arithmetic assignment statements from one high level programming language into another, as provided by most computer algebra sytems as a simple output option. One aspect is decomposition and vectorization. Another is code generation, code optimization and, if possible, *a priori* error analysis. Roughly speaking the first category is strongly related to problem class and target machine, while the latter group has a more problem independent flavor. Instead of problem classes it is perhaps better to think of classes of solution techniques. Examples are solution strategies for elliptic partial differential equations, finite element analysis techniques, Newton-Raphson methods, requiring (inverse) Jacobians, optimization techniques demanding Hessians, etc. We developed code for Jacobian and Hessian production in a NAG-library compatible way [12,18]. This package, shortly indicated as GENJAC, is written in RLISP as an extension of REDUCE, one of the leading computer algebra systems [4,9]. This experimental package is based on the use of some of the problem independent tools of the symbolic-numeric interface, i.e. GENTRAN [5,6,7] and SCOPE

[14,15]. It serves as a starting point for further research, dedicated to decomposition, vectorization and automated error analysis strategies. This "learning from the concrete" is of interest for the solution of many practical problems, ranging from neural networks to maximum likelihood estimations in econometry. In addition, it may be a valuable experience for other classes of solution strategies. Our intention is to explain the present status of GENJAC, i.e. version 2, and its role in our (future) research in sections 3 and 4. We first give a short introduction to GENTRAN and SCOPE in section 2.

2. GENTRAN and SCOPE illustrated. GENTRAN allows to translate syntactically correct MACSYMA [19] or REDUCE commands, or their evaluations (including function definitions, declarations and the like) into their equivalent in FORTRAN, RATFOR or C. SCOPE is designed to assist in minimizing the arithmetic complexity of computer algebra output. This output consists often of lengthy expressions, grouped together in blocks of straightline code for numerical execution. The optimization strategy is based on heuristic techniques for finding, and adequately replacing, common subexpressions in such blocks of code.
As an example we discuss the production of optimized FORTRAN-code, using both GENTRAN and SCOPE, for computing the inverse of the symmetric (3,3)-matrix M, given in Figure 1. The same example was earlier discussed in [5].

```
                  2                          2                  2                       2
M(1,1) := - ((9*P *M30 + J30Y - J30Z)*SIN(Q3) - (18*M30 + M10)*P  - 18*COS(Q3)*COS(Q2)*P *M30 - J30Y - J1OY)

                           2                           2                       2          2
M(2,1) := M(1,2) := - ((9*P *M30 + J30Y - J30Z)*SIN(Q3)  - 9*COS(Q3)*COS(Q2)*P *M30 - 9*P *M30 - J30Y)

                                         2
M(3,1) := M(1,3) := - 9*SIN(Q3)*SIN(Q2)*P *M30

                2                           2      2
M(2,2) = - ((9*P *M30 + J30Y - J30Z)*SIN(Q3)  - 9*P *M30 - J30Y)

M(3,2) := M(2,3) := 0

             2
M(3,3) := 9*P *M30 + J30X
```

Figure 1: *The matrix M*

Gaussian elimination, and hence determinant calculation, is quite efficient in a numerical context. The space complexity is $O(n^2)$ and the time complexity is $O(n^3)$. Both are different in a computer algebra setting [10], and strongly related to the structure and size of the matrix entries and of course also to the size of the matrix itself and its degree of sparsity. Although computation of the inverse MNV of the matrix M is seemingly simple (a command like $MNV := M^{-1}$ suffices) each entry of MNV is a rational expression with $det(M)$ as its denominator. The entry $MNV(1,1)$, thus obtained,is:

```
                                                           2           2
MNV(1,1) := ((9*(J30Y - J30Z + J30X)*P *M30 + 81*P *M30  + J30Y*J30X - J30Z*J30X)*SIN(Q3)  - 9*(J30Y + J30X)*P *M30

     4  2
- 81*P *M30  - J30Y*J30X)/(((9*((J30Y - J30Z)*(J1OY + J30X) +  J1OY*J30X)*M30 + J30Y*M10*J30X - J30Z*M10*J30X)*P

                                             4          2            2 4            2
+ 9*(9*(J30Y - J30Z + J1OY + J30X)*M30 + (J30Y - J30Z + J30X)*M10)*P *M30 + 81*(9*P *M30 + J30Y)*SIN(Q2) *P *M30

              6     2                      2
+ 81*(9*M30 + M10)*P *M30  + J30Y*J1OY*J30X - J30Z*J1OY*J30X)*SIN(Q3) - (9*((J1OY + J30X)*J30Y + J1OY*J30X)*M30 + J30Y*M10*

   2                  2         4                2    4   2
J30X)*P  - 81*((J30Y - J30Z) + 9*P *M30)*SIN(Q3) *SIN(Q2) *P *M30  - 9*(9*(J30Y + J1OY + J30X)*M30 + ( J30Y + J30X)*M10)*

4            2              2        2 4         2                 6   2
P *M30 + 81*(9*P *M30 + J30X)* COS(Q3) *COS(Q2) *P *M30  - 81*(9*M30 + M10)*P *M30  - J30Y*J1OY*J30X)
```

When straightforwardly translating the results of formal algebraic processing into numerical code many inefficiencies occur. Code for computing the non-zero elements of MNV can contain many $det(M)$ computations. An alternative is to compute MNV numerically. But then tools are required, as part of the symbolic-numeric interface, to determine a priori and automatically the precision required to perform these numerical computation in a reliable fashion. Work in this direction is also in progress [13,17]. A combined use of GENTRAN and SCOPE provides a simple mechanism for the construction of a fairly optimal description of MNV, as illustrated in Figure 2.

```
      REAL Q3,Q2,M30,P,J30Y,J30Z,J10Y,M10,J30X,A1,A13,A11,A10,A19,A17,
     . J1OY,A18,M(3,3),T0,T1,T2,T3,T4,B21,B12,B3,MNV(3,3)

      A1=SIN(REAL(Q3))
      A13=P*P
      A11=A13*M30
      A10=A11*COS(REAL(Q2))*COS(REAL(Q3))
      A19=9*A11
      A17=(A19+J30Y-J30Z)*A1*A1
      M(1,1)=18*A10+J30Y+J1OY+A13*(18*M30+M10)-A17
      A18=A19+J30Y-A17
      M(1,2)=A18+9*A10
      M(1,3)=-(A19*SIN(REAL(Q2))*A1)
      M(2,?)=A18
      M(2,3)=0
      M(3,3)=A19+J30X

      T0=M(1,1)
      T1=M(1,2)
      T2=M(1,3)
      T3=M(2,2)
      T4=M(3,3)

      B21=T1*T1
      B12=T0*T4-(T2*T2)
      B3=B12*T3-(B21*T4)
      MNV(1,1)=(T4*T3)/B3
      MNV(1,2)=-(T1*T4)/B3
      MNV(1,3)=-(T2*T3)/B3
      MNV(2,2)=B12/B3
      MNV(2,3)=(T2*T1)/B3
      MNV(3,3)=(T0*T3-B21)/B3

      DO 25001 J=1,3
         DO 25002 K=J,3
            M(K,J)=M(J,K)
            MNV(K,J)=MNV(J,K)
25002    CONTINUE
25001 CONTINUE
```

Figure 2: *The code generated for the computation of MNV*

The code presented in Figure 2 consists of 5 sections; it was made on line with a mixture of REDUCE, GENTRAN and SCOPE commands. The declarations are followed by a section defining the computation of the relevant entries of M. Then some lines are used to rename the entries; these new identifiers are temporary variables used in the computation of MNV. The fourth section defines the computation of the relevant entries of MNV. It shows the essential role played by the placeholders. A double loop is finally used to obtain values for the under-triangular entries of both M and MNV. The description of the M-entries shows common subexpression (cse) definitions and uses. The cse-names start with the letter A. The MNV-section has a similar structure; this time the cse-names start with a B. Both sections were considered as blocks of code and were optimized seperately, using SCOPE. Hence both sections show system-generated cse names, which are also placed in the declaration section. The declarations, the temporary variables and the loop-construction are all obtained by using special GENTRAN features.

As suggested by the example it is possible to interactively construct programs, mainly consisting of extensive, but optimized arithmetic. It also shows that declaration production can be automatized as well. However, it demands a lot of user intervention and activities, besides working knowledge of some of the internal features of a number of software facilities, applicable as extensions of REDUCE. And, we only demonstrated the production of a piece of sequential code.

Since determinant calculation in a computer algebra setting is an exponential process, the size of the matrix we have to deal with, can not be too large. So, when a inverse Jacobian is required for some computation it is best not to attempt to use a computer algebra system for it. Automatic generation of a Jacobian or Hessian however can be done quite efficiently, as discussed in the next session.

3. Generation of Jacobians and Hessians. GENJAC.1 [12] was based on a combined, but restricted use of both GENTRAN and SCOPE. We concentrated on the production of reasonably efficient code for computations involving a system of non-linear equations and the associated Jacobian. Optimization, using SCOPE, was performed blockwise , like shown in the example of the previous section. The Hessian production facility was simply added by programming a few extra lines, but was limited in its capacities.

GENJAC.2 [1] is our present facility. It is also coded in RLISP as an extension of REDUCE, like GENTRAN and SCOPE. The program accepts a description D_E of a set of of nonlinear equations E. Specification of C or Pascal as a target language is possible. Fortran is the default selection. The user can request the construction of programs P_E, P_J, P_H or combinations of such as P_{EJ}, P_{EJH}, for instance in the form of procedures, which are compatible with relevant NAG [18] library routines. The programs are all globally optimized. P_E defines the computation of E. P_J and P_H are similar definitions for the Jacobian of E and the Hessian of E, assuming $|E| = 1$, respectively.

Let us first discuss how to obtain P_E, P_J or P_{EJ}. The construction of P_H or P_{EJH} is based on a similar but repeatedly applied approach. GENJAC.2's use is straightforward and syntax driven. Four different commands can be used for the construction of D_E, depending on the structure of E: **eqvars**, **eqindices**, **puteq** and **constraints**, as shown in Figure 3.a.

```
%GENJAC.2's input:

 eqvars a,b$

 eqindices j,k$

 puteq (a(j,k)-b(j,k))*a(j,k)-sin(eps)
      index j,k from 1,1 upto 3.3$

 puteq (a(j,k)+b(j,k))*a(j,k)-sin(eps)
      index j,k from 1.1 upto 3.3$

 mapfiles fw,bw$

 genfdf foo header func,x,f,jac$
```

a

```
            SUBROUTINE FUNC(X,F,JAC)
        INTEGER I1,I2,J,K,NDIM,G43,G33,G36
           REAL X(18),EPS,G37,G44,F(18),JAC(18.18)
C          PLEASE NOTE "
  C         THE FOLLOWING VARIABLES ARE GLOBAL
C       EPS
             NDIM=18
             DO 25001 I1=1,18
          DO 25002 I2=1,18
                  JAC(I1,I2)=0
         25002     CONTINUE
25001 CONTINUE
            G37=SIN(EPS)
        DO 25003 J=1,3
                DO 25004 K=1,3
              G43=3*J+K
              G33=G43-3
              G36=G43+6
              F(G33)=X(G33)*(X(G33)-X(G36))-G37
              JAC(G33,G36)=-X(G33)
              G44=2*X(G33)
              JAC(G33,G33)=G44-X(G36)
              F(G36)=X(G33)*(X(G33)+X(G36))-G37
              JAC(G36,G36)=X(G33)
              JAC(G36,G33)=G44+X(G36)
          25004     CONTINUE
          25003 CONTINUE
              RETURN
              END
```

b

Figure 3: *a. Input, b. Resulting code*

It also shows that some other commands are available for code production. These commands are discussed below. It is permitted to use subscripted independent variables in the definition of the different equations. This can lead to classes of equations and thus also to constraints. Let us assume, as an example, that E is formed by the following sets of equations:

$$\begin{cases} E_1 & = & \{(a(j,k)-b(j,k))*a(j,k)=sin(\epsilon)|1\leq j\leq 3, 1\leq k\leq 3\} \\ E_2 & = & \{(a(j,k)+b(j,k))*a(j,k)=sin(\epsilon)|1\leq j\leq 3, 1\leq k\leq 3\} \end{cases}$$

REDUCE is informed with
eqvars a,b$
that a and b -both possibly subscripted- function as independent variable names. The command
eqindices j,k$
states in addition that j and k will function as indices, when ever required. The classes E_1 and E_2 are introduced using the commands

```
puteq     (a(j,k)+b(j,k))*a(j,k)-sin(eps) index j,k from 1,1 upto 3,3$
puteq     (a(j,k)-b(j,k))*a(j,k)-sin(eps) index j,k from 1,1 upto 3,3$
```

These descriptions are internally stored in the form of records. Initially this set of records $\mathcal{R}$ is in fact unordered, since any input order for the elements of E is admissible. Each record R_i defines one class of equations E_i. R_i is a quadruple (D_i,V_i,S_i,L_i). D_i is the internal representation of the expression characterizing E_i. V_i is the set of independent variables, used in the definition of E_i and S_i the corresponding set of subscript ranges. L_i finally is an expression, to be used to localize the rows in the Jacobian matrix holding the partial derivatives of D_i w.r.t. the elements of V_i.

A record representation can be:

D_i		
V_i	S_i	L_i

The record generated for the equation class E_1 has the form as shown in Figure 4.a

$(a(j,k)-b(j,k))*a(j,k)-sin(\epsilon)$		
$(a(j,k)\ b(j,k))$	$((1\ 3)\ (1\ 3))$	$3*j+k-3$

a

```
FOR J:=1:3 DO
FOR K:=1:3 DO
<<FF($L_1$) := $D_1$;
JJ($L_1$,col$_a$):=$\frac{\partial D_1}{\partial a(j,k)}$;
JJ($L_1$,col$_b$):=$\frac{\partial D_1}{\partial b(j,k)}$>>
```

b

Figure 4: *a. Record, b.Generated code*

The content of D_1, V_1 and S_1 is obvious. Assuming E_1 is the first equation class we have been entering, the first 9 rows of the Jacobian matrix will be required to store the gradients of the elements $f_{1,j,k}$ of E_1. Since k is associated with the inner loop and j with the outer the row $3*j+k-3$ of the Jacobian matrix is used to store the $\nabla f_{1,j,k}$. It may happen in some cases -due to boundary conditions, for instance- that some of the independent variables have initial values. These constrained variables can be made known to the system with a special command. A constraint like $C: a(1,1)=1$ is introduced to the system with the command
constraints a(1,1)=1$
Such a constraint can influence the contents of the different components of some of the records, requiring some redefinitions. It can even lead to an enlarged initial record set $\mathcal{R}$. When, for instance applying C on R_1 three new records are required (Figure 5).

The thus extended set $\mathcal{R}$ is in fact still unordered. But constraint-based record modifications guarantee that all elements of $\mathcal{R}$ can either be translated into a (nested) loop or into an assignment statement. Once input processing is completed RLISP code C_i is made for each record R_i. C_i consists of atmost three components. A loop header made out of the S_i information, the definition of an assignment statement, defining how to compute D_i and finally

a set of partial derivatives, being the non-zero entries of the Jacobian matrix for this R_i, using its D_i, V_i and L_i information. Figure 4.b shows the code generated for E_1. Once the transformation $\mathcal{R} \Rightarrow \mathcal{C}$ is made it is possible to apply GENTRAN directly to this intermediate rough form, thus obtaining equivalent code in the target language.

<table>
<tr><td colspan="3">$1 - b(1,1) - sin(\epsilon)$</td></tr>
<tr><td>$(b(1,1))$</td><td>[(- -) (- -)]</td><td>1</td></tr>
</table>

a

<table>
<tr><td colspan="3">$(a(j,k) - b(j,k)) * a(j,k) - sin(\epsilon)$</td></tr>
<tr><td>$(a(j,k)\ \ b(j,k))$</td><td>[(2 3) (1 3)]</td><td>$3 * j + k - 3$</td></tr>
</table>

b

<table>
<tr><td colspan="3">$(a(1,k) - b(1,k)) * a(1,k) - sin(\epsilon)$</td></tr>
<tr><td>$(a(1,k)\ \ b(1,k))$</td><td>[(- -) (2 3)]</td><td>k</td></tr>
</table>

c

Figure 5: *Added records*

We can even apply SCOPE to locally optimize the different blocks of code, given by the C_i, and as illustrated in section 2. But we can do better. The set $\mathcal{C}$ can be ordered by applying some transformations -loop fusion, global code optimization and code motion- based on the result of a data dependence analysis. We extended GENTRAN with a data dependence analyser DDA [11]. This module accepts programs in the intermediate code, such as $\mathcal{C}$, which are constructed with arithmetic assignment statements and loop structures only. Statements S_1 and S_2 are called data dependent if both statements refer to the same identifier V. Control dependence occurs when a loop header index is refered by a statement. In view of the structure of $\mathcal{R}$ and thus of $\mathcal{C}$ we only expect (control) dependences in the intermediate code $\mathcal{C}$. Subjecting $\mathcal{C}$ to a DDA-search produces relevant information about control dependences and thus about the real execution order of the C_i, together forming $\mathcal{C}$. We associate a set **execord** with $\mathcal{C}$, defining the execution order of the (sub)sections of the C_i's.

Once all control dependences are known **loop fusion** becomes possible. All R_i were transformed into C_i, implying that identical loop headers may have been created. Assume we use the indices $i_1, i_2, \ldots, i_n$. If for E_1 and E_2 an m exists such that the ranges of i_j in both E_1 and E_2 are equal, $1 \leq j \leq m$, the loop structures for E_1 and E_2 can be combined together:

```
FOR $i_1$:=$a_1$ : $b_1$ DO
FOR $i_2$:=$a_2$ : $b_2$ DO
        ⋮
FOR $i_m$:=$a_m$ : $b_m$ DO
<<
    <Assignment Block for $E_1$ >;
    <Assignment Block for $E_2$ >
>>
```

All bookkeeping required to actually accomplish the possible fusions, i.e. code contractions, is done by modifying the content of the elements of **execord**. Hence **execord** possibly induces a permutation of the original execution order.Once the final execution order of the C_i sections or fused sections of them is determined, **global code optimization** can be performed. Optimization using SCOPE means removal of redundant arithmetic with fast heuristic techniques. Only minor changes in the internal data handling structures were required to incorporate data dependence considerations in the heuristics of SCOPE. Hence all arithmetic defining portions of code can be collected and analysed. However, the limited set of indices used for the description of the arithmetic assignment statements can lead to erroneous results. Identical expressions in indices, but occuring in different sections of the intermediate code can not be considered as identical computations. To avoid this, each loop header left after the process of loop fusion receives its own unique iteration identifier. These indices are also substituted in the other sections of the loop, i.e. in the arithmetic expressions, which are going to be optimized. Once optimization is performed, in fact a transformation of the blocks of arithmetic occuring in $\mathcal{C}$ and using SCOPE, $\mathcal{C}$ has to be restructured to take these improvements into account, again leading to some modifications of **execord**. The now created version of C consists of loops and blocks of assignment statements. Due to the strategy, outlined so far, it may happen that common subexpressions are nested too deeply, simply because they are defined with less indices than required at that level, or without any index at all. This was an additional reason to apply the DDA-module. We apply **code motion** to move all such loop invariant computations outside the loops containing them. This is accomplished by a further modification of **execord**. Once these transformations are completed we can produce the programs P_E, P_J or P_{EJ}, using GENTRAN.

Figure 3 b) gives P_{EJ} for the set E, used as illustration so far. It shows that loop fusion is performed, that some code optimization was possible and that the code for computing $sin(\epsilon)$ was moved outside the loop. Some commands can be used to influence code production, as also shown in Figure 3.a

The command **genfdf** FOO$ results in a complete program P_{EJ} stored in file FOO with by default names in the header for the independent variables (XX), the definition of the equations, with right hand sides reduced to 0 (FF) and for the entries of the Jacobian (JJ). However the **genfdf** command can optionally be extended with two features. The **split** FOO1 option gives P_E in file FOO and P_J in file FOO1. The second option, further extending the command, is **header** FUNC(X,F,J). Here FUNC, X, F and J denote user selected names for P_E, P_J (DFUNC) and XX, FF and JJ, respectively. In addition execution of the command **mapfiles** FW,BW$ leads to a procedure, defining a forward map of the independent variables on XX or its equivalent, stored in the file FW and to a similar backward facility stored in file BW. The content of FW for our example is:

```
      SUBROUTINE FORWMAP(A,B,XX)
      REAL XX(18),A(3,3),B(3,3)
      INTEGER G0050,G0051,G0052,G0053
      DO 25007 G0050=1,3
         DO 25008 G0051=1,3
            XX(3*G0050+G0051-3)=A(G0050,G0051)
25008    CONTINUE
25007 CONTINUE
      DO 25009 G0052=1,3
         DO 25010 G0053=1,3
            XX(3*G0052+G0053+6)=B(G0052,G0053)
25010    CONTINUE
25009 CONTINUE
      RETURN
      END
```

The strategy, outlined so far for the production of Jacobian code, requires only minor modifications for production of P_{EJH} for the Hessian matrix H, associated with some function f. Like done for P_{EJ} we start setting up a set $\mathcal{R}'$ of records R'_i, using the **eqvars**, **eqindices** and **puteq** commands. Assuming $f = f(x_1, x_2, \ldots, x_n)$, the D'_i component of R'_i is $\frac{\partial f}{\partial x_i}$, the component V''_i is $(x_1, x_2, \ldots, x_n)$, the S'_i component is not required and the L'_i component is simply i. An important difference with the above outlined strategy has to be remarked. H is symmetric. since $\frac{\partial^2 f}{\partial x_j \partial x_i} = \frac{\partial 2 f}{\partial x_i \partial x_j}$. Therefore we do not produce the under-triangular part of H, but generate like shown in the example in section 2, a double loop for this part of H. The subroutine below shows P_{EJH} for $f = f(x,y) = x^2 * y^2$.

```
      SUBROUTINE FUNC(XX,FF,JJ,HH)
      INTEGER I1,I2,NDIM
      REAL XX(2),G41,FF,G43,JJ(2),HH(2,2)
      DO 25001 I1=1,2
         DO 25002 I2=1,2
            HH(I1,I2)=0
25002    CONTINUE
25001 CONTINUE
      G41=XX(2)*XX(1)
      FF=G41*G41
      G43=2*G41
      JJ(1)=G43*XX(2)
      HH(1,1)=2*XX(2)*XX(2)
      JJ(2)=G43*XX(1)
      HH(2,1)=4*G41
      HH(2,2)=2*XX(1)*XX(1)
      DO 25003 I1=1,1
         DO 25004 I2=I1+1,2
            HH(I1,I2)=HH(I2,I1)
25004    CONTINUE
```

```
25003 CONTINUE
      RETURN
      END
```

4. Conclusions and Prospects. We briefly indicated the present state of our research concerning automated generation of reliable and efficient numerical code. Our goal is to develop tools, which are easily usable in an adequate programming environment and which only require problem specification and definition of target hardware and sofware for the production of a reliable solution.

We decided to employ automated generation of Jacobians and Hessians not only to gain experience in the diversity of tasks, required for such an operation, but also because many interesting applications are known and provide a practical setting for feed-back. Decomposition -or phrasing it slightly differently process distribution- is probably not too complicated and do-able like vectorization of processor code. We already performed some quite satisfactory experiments with code vectorization [8]. These experiments were partly based on a modified use of some parts of the SCOPE code. It stimulated further research in this direction [21]. It also made evident that the introduction of flexible vector- and tensor operations in REDUCE is a need. We intend to use these facilities also to further improve our code for Hessian production.

We also decided to reconsider the design of GENTRAN. GENTRAN was originally made for the generation of code for the traditional von Neumann architectures. But it now lacks features to assist in generating programs for alternative architectures. This has lead to the development of GENCRAY [23] and GENW2 [20]. Both packages are used for the production of finite element code, to be executed on a CRAY- or a Warp-machine, respectively. The output of GENCRAY is FORTRAN-77 or CRAY FORTRAN-77, while GENW2 is a W2-code generator. W2 is a PASCAL-like high level programming language for the Warp array. These variations on the GENTRAN-line form an answer to the question how to use a workstation as a front-end. But is it an adequate answer?

GENTRAN is a translation tool. Hence -when really using GENTRAN and thus REDUCE with more general intentions- a more profound approach is required. A translation into a parallel loop, likewise demands the existence of a syntactical construction to define this parallel operation. Hence, not only GENTRAN has to be modified, but REDUCE as well. This is at present being investigated as well [16].

References

[1] Berger, F.C.: "Automatic generation of Jacobian routines", Masters Thesis, Dept. of Comp. Sc., Univ. of Twente (march 1991).

[2] A. Boyle, B.F. Caviness (eds): "Future directions for research in symbolic computing", Philadelphia: SIAM (1990).

[3] Doleh, Y. and Wang, P.S.: "SUI: A system independent user interface for an integrated scientific computing environment", Proceedings ISSAC '90 (S. Watanabe and M. Nagata, eds), 88-95. New York: ACM-Press (1990).

[4] Fitch, J.: "Solving algebraic problems with REDUCE", J. Symbolic Comput. 1, 211-238 (1985).

[5] Gates, B.L.: "GENTRAN: An automatic code generation facility for REDUCE", ACM SIGSAM Bulletin 75, 24-85 (1985).

[6] Gates, B.L.: "A numerical code generation facility for REDUCE", Proceedings SYMSAC '86 (B.W. Char, ed.), 94-99. New York: ACM-Press (1986).

[7] Gates, B.L. and Wang, P.S.: "A LISP-based RATFOR code generator", Proceedings 1984 MACSYMA User's Conference. (V.E. Golden, ed.), 319-329. Schenectady: G.E. (1984).

[8] Goldman, V.V. and van Hulzen, J.A.: "Automatic code vectorization of arithmetic expressions by bottom-up structure recognition", in "Computer Algebra and Parallelism" (J.Dela Dora and J. Fitch, eds), 119-132. London: Academic Press (1989).

[9] Hearn, A.C.: "REDUCE user's manual", version 3.3. Santa Monica, CA: The Rand Corporation (1987).

[10] Horowitz, E. and Sahni, S.: "On computing the exact determinant of matrices with polynomial entries", Journ. ACM 22, 38-50 (1975).

[11] van Heerwaarden, M.C.: "Data dependences analysis for program generation and optimization", Masters Thesis, Dept. of Computer Science, Univ. of Twente (june 1989).

[12] van den Heuvel, P. ,Goldman, V.V. and van Hulzen, J.A. : "Automatic generation of FORTRAN-coded Jacobians and Hessians", Proceedings EUROCAL '87 (J.H. Davenport, ed.), Springer LNCS series nr 378, 120-1[illegible]. Heidelberg: Springer Verlag (1989).

[13] Hulshof, B.J.A. and van Hulzen, J.A.: "Automatic error cumulation control", Proceedings EUROSAM '84 (J. Fitch, ed.), Springer LNCS series 174, 260-271. Heidelberg: Springer Verlag (1984).

[14] van Hulzen, J.A., e.a.: "SCOPE 1, A Source-Code Optimazation PackagE for REDUCE - User's Manual", Dept of Comp. Science, Univ. of Twente (In preparation).

[15] van Hulzen, J.A., Hulshof, B.J.A., Gates, B.L. and van Heerwaarden, M.C.: "A code optimization package for REDUCE", Proceedings ISSAC'89 (G. H. Gonnet, ed.), 163-170 New York: ACM-Press (1989).

[16] Jongerius, E.P.: "PARTRAN - A parallel version of GENTRAN", Masters Thesis, Dept. of Comp. Science, Univ. of Twente (june 1991).

[17] Molenkamp, J.H.J., Goldman, V.V. and van Hulzen, J.A.: "An improved approach to automatic error cumulation control" , Proceedings ISSAC '91 (to appear).

[18] NAG FORTRAN Library Manual Mark 13. Numerical Algorithm Group, Oxford (1988).

[19] Pavelle, R. and Wang, P.S.: "MACSYMA from F to G", J. Symbol. Comput. 1, 69-100 (1985).

[20] Sharma, N. and Wang, P.S.: "Symbolic derivation and automatic generation of parallel routines for finite element analysis", Proceedings ISSAC '88 (P. Gianni, ed.), Springer LNCS-series nr 358, 33-56. Heidelberg: Springer Verlag (1989).

[21] Verheul, J.C.: "Structure Recognition", Masters Thesis, Dept. of Comp. Science, Univ. of Twente (march 1991).

[22] Wang, P.S.: "A system independent graphing package for mathematical functions", Proceedings DISCO '90 (A. Miola, ed.), Springer LNCS series nr 429, 245-254. Heidelberg: Springer Verlag (1990).

[23] Weerawarana, S. and Wang, P.S.: "GENCRAY: A portable code generator for Cray Fortran", Proceedings ISSAC'89 (G.H. Gonnet, ed.), 186-191. New York: ACM-Press (1989).

About the Work of the Committee on the Software Packages for Mathematical Physics and the Committee of the Programming Technology and Software Tools for the Computational Experiment

by

Yu. I. Shokin
Institute of Computational Technologies
Siberian Division of the USSR Academy of Sciences
Novosibirsk 630090
USSR

The Committee on software packages for mathematical physics is a non-governmental society of researchers and experts working on creation of new computational algorithms for the problem solution of mechanics of continuous medium, designing general-purpose and special-purpose computers for the solution of "cumbersome" problems of mathematical physics as well as on the organization and carrying out of computational experiments in hydro- and gas dynamics, mechanics of solids under strain etc.

This Committee is part of the Section of the software for the mathematical modelling of the Scientific Council on the complex project "Mathematical Modelling" of the USSR Academy of Sciences.

This Committee works in close cooperation with the Committee of the programming technologies and software tools for the computational experiment directed by V.I. Legon'kov (Tchelyabinsk).

Those both Committees were organized on the initiative of the late Academician Yanenko who was the chairman of the First Conference on the problems of cumbersome problems solution by computers held in Novosibirsk in 1971. The second conference (Moscow province, 1972) defined the scope of the problems under consideration, i.e. the computer realization of numerical methods of problem solution of mathematical physics, operating and language facilities for different stages of the development and operation of the applied programs and analytical computations.

Further on, the conferences became traditional; the eighth conferences (the last one presided by Academician N.N. Yanenko) was held in Tashkent in 1983.

Since 1986 the Committee on the software packages for mathematical physics has been headed by the corresponding member of the USSR Academy of Sciences Yu. I. Shokin and the Conference on the software packages for the mathematical physics resumed its regular work (once every two years) which had been interrupted by the death of Academician Yanenko. In the time between the conferences working sessions of the Committee were held (together with the Committee headed by V.I. Legan'kov). As a rule, such sessions were held after the all-Union Conference of young scientists on the numerical methods in the mechanics of continuous medium.

In 1987 the working session of the Committee on the software packages for mathematical physics tool place which determined the principal directions of research, forms of activity and the personal membership.

The participants decided to concentrate their efforts on

- regular and timely exchange of information on the development of computational methods of mathematical physics and their implementation in programs, creation of application packages, and mastering new models of computers.
- as well as on the coordination of projects of general system and applied software and in the program realization of computational algorithms

- the mapping of the computational methods' structure onto the architecture of computers
- statement of the requirements to the program implementation of new efficient numerical models and algorithms with the aim of working out the standards of documentation and publication of the time-and space consuming programs of mathematical physics;
- on computer efficiency in problem solution in the mechanics of continuous medium.

It was suggested that the members of the Committee should write summaries on the subjects within the scope of the Committee activities including:

- Program realization of the algorithms of mathematical physics on the computer systems of modern architecture;
- Perspective development of computers in the USSR and overseas;
- Basic structures of the groups specialized in computational experiments in natural science;
- General system software at present and its prospective development;
- Reviews of Proceedings of all-Union and international symposia and conferences.

Among the members of the Committee are professor Yu. I. Shokin, professor E.A. Bondarev (Yakutsk), professor V.P. Ilyin (Novosibirsk), professor V.A. Karlov (Moscow), professor V.F. Kuropatenko (Tchelyabinsk), professor A.F. Sidorov (Sverdlovsk), professor A.N. Shevtchenko (Kharkov), professor I.A. Nikolayev (Rostov-na-Donu) et al.

In 1986-1990 the Committee held five working sessions, three all-Union Conferences on the software complexes for mathematical physics, organized the publication of two volumes of proceedings and several preprints.

The scientific program of the Committee included:

- Models of multicomponent turbulized media;
- Models of the mixing of heterogeneous media (V.F. Kuropatenko);
- Algorithms and software for the problem solution of chemical kinetics (Ye.A. Novikov):
- Algorithms and software for the problems of resistivity prospecting (S.M. Bersenev);
- Method of incomplete factorization, new results (V.P. Ilyin);
- Employment of R-functions in problem solution of mathematical physics (G.P. Man'ko);
- Modification of the method of finite volume for axiosymmetrical problems of gas dynamics (S.N. Martyushov);
- Peculiarities of mathematical modelling of non-isothermal gas filtration (E.A. Bondarev);
- Numerical modelling of non-equilibrium processes in channels of variable cross-sections (Yu. N. Deryugin);
- Qualitative properties of the solution of equations with sign-changing viscosity (V. A. Novikov);
- On some semi-analytical methods of solution of non-linear problems of the mechanics of continuous medium (A.F. Sidorov);
- Method of differential approximation employed for the qualitative analysis of difference circuits schemes (Yu. I. Shokin)

In cooperation with the Committee for programming technologies and software tools for computational experiment general methodological problems have been discussed, such as:

- Problems of software development for computational experiment (V.I. Legon'kov);
- Vector-conveyor computer system (programming of problems of mathematical physics) (L.N. Stolyarov);
- Estimation of level and quality of software facilities (A.V. Vladytskiy);
- Architecture of high-performance computers;
- Computational experiment under conditions of mass computations by different user groups (L.V. Nesterenko);
- Organization of mathematical departments engaged in computational experiment (V.F. Kuropatenko).

In addition, some already developed software systems have been discussed:

- SAFRA – support system for creation and operating the application packages for mathematical physics (Institute of Applied Mathematics, Moscow)

- Automatic SATURN-technology of the design and support of program complexes (dialogue application packages) with automatic scheduling and flexible monitoring of the computation process (Irkutsk computing centre);
- Software tools for support and methods of program development in high-energy physics (International Institute of Nuclear Research Dubna);
- MOST-technology (Institute of Technical Physics, Tchelyabinsk);

Committee of programming technology and software tools for computational experiment worked on the problems associated with the development of powerful application software, packages and program complexes for the computational experiments for solution of equations with partial derivatives. The members of this Committee were representatives of both Academy of sciences and industry as well as from different computing centres. The Committee is headed by V.I. Legon'kov a recognized authority in this subject.

The aim of Committee was to develop general requirements to the software for computational experiment which on the one hand should be based on the international standards of software technology and on the other hand could employ the problem-oriented technologies already developed in our country. In 1987-1991 the Committee mostly dealt with:

- problem-oriented development technologies of the powerful programs of mathematical physics;
- estimation of application programs;
- development of the model of labour consumption of a program.

The development of the problem-oriented technologies and software tools for the development of powerful programs intended for the solution of equations with partial derivatives is one of the most popular trends in this field in the Soviet Union. One of the first projects in this sphere is the SAFRA system created by a group headed by professor A.A. Samarskiy. This system has much in common with the well-known OLYMPUS system.

Later, many systems have been developed by various organizations in the USSR based on fairly original but sometimes contradictory concepts. Among them we could mention systems intended for the solution of differential equations: SATURN developed in Irkutsk Computing Centre under the supervision of academician V.V. Matrosov and the software tools SOK and SOP developed by a team headed by V.I. Legon'kov. All these systems are fairly advanced, employ language facilities of their own and have a set of system components supporting application technology when application programs are being developed. Nevertheless, they are too dissimilar, indeed, no coordinated use of the programs developed by means of different software tools is possible because of the difference in the organization of the supporting systems.

Several meetings of the Committee were devoted to the analysis and comparison of the systems SATURN, SOK and SON. The authors of the project also took an active part in the discussion. The resulting recommendations defined the most promising application for every system. It must be noted that although application characteristics are usually included into the user's manual for every system the document developed at the meetings was primarily based on the comparative analysis.

Some organizations in the West (e.g. Electricité de France) also perform regular comparisons of different software tools in order to work out the recommendations for their application, this work seems very useful not only practically but also methodologically.

Let us dwell on the problems of the qualitative estimation of programs. The objective set of criteria for such an estimation is an important part of any program production technology if it is intended for the practical use. From our viewpoint, the most interesting thing is not to estimate an isolated program but to evaluate a set of programs, especially a sequence of versions of one and the same program. Practically, the importance of such estimation is due to the possibility of control over the processes of the development and optimization of a software tool. This, rather pragmatic, viewpoint brought us to the revaluation of many well-known evaluation methods of the program quality. As the basic initial solution variants, the Committee considered both the works of Western experts (B. Boem, G. Meiers, Van Tassel, J. Fox) and the Soviet programmers (V. Lipayev, V.I. Legon'kov, A.F. Kulakov). After discussion the members of the Committee worked out the recommendations for the analysis and quality estimation of the programs of mathematical physics. These recommendations included a system of criteria so that the estimation could be presented either as an index vector or as a weight formula with a certain system of weights.

As for the model of the labour consumption of the software tool development, no common point of view has been developed though this problem has been being discussed since 1989. As a matter of fact, all the currently used models of this kind are far from being realistic. They usually take into consideration only one resource required for the program development, i.e. strength of the personnel. Such is, for instance, the model of Barry Boem. In practice, we are, however, interested in the whole range of the consumed resources which must include at least:

- computational resource which comprises a set of various computational facilities employed in development and operation of a software tool;
- human(labour) resource, i.e. the number and qualification of the team of designers;
- the auxiliary (overhead) resource including different indirect costs (the rent, electricity bills, wages of service staff etc.)

For some of these resources specific time dependencies are known over the life cycle of a program. However, in our opinion, an integrated model of labour consumption must be developed comprising the most complete set of estimates and their functional interrelations including both static and dynamic dependencies.

As different models of personal computers are now widely used in numeric modelling the Committee had to consider the PC employment in the computational experiment and the efficiency of the development and use of the general system and software tools for this kind of computers and also the necessity to arrange exhibitions demonstrating new software systems.

At the last Conference (1990, Rostov-na-Donu) it was pointed out that in the USSR there were up-to-date algorithms of mathematical physics and adequate program facilities that can prove viable both on the national and international market. It mostly refers to the program modules realizing new algorithms of mathematical physics.

We think it very important and useful that the experts from different countries working within the IFIP 2.5 group should be informed about the results of their Soviet colleagues. To this end, we could translate the Proceedings of the Committees' conferences into English and present the reviews and prospectuses of specific program systems.

In conclusion, it must be noted that the Committee on program complexes is planning to hold its next Conference in Ulan-Ude, 1992 and to publish its Proceedings. New algorithms and programs of mathematical physics will be discussed, the development trends of the computational mathematics, computer engineering and programming methods will be studied. Development and use of the intellectualization systems in computational experiment will be investigated as well as the program development technologies for the mathematical physics and computational experiments. The Committee is going to encourage the market research and the marketing prospects for the application software.

Mathematical User Interfaces for Graphical Workstations

by

Allan Bonadio
Prescience Corporation
939 Howard St.
San Francisco
CA 94103
USA
bonadio@well.sf.ca.us

Introduction

This paper explores the user interface issues that are important for the future of scientific computing. First, user interfaces in general are explored. Next, the focus is narrowed to scientific applications. Finally, there is a survey of some user interface techniques that I think are of interest.

Disclaimer: The author is president of Prescience Corporation which develops and publishes two commercial packages mentioned: Expressionist and Theorist.

User Interfaces

The topic of user interface is popular these days, although frequently there is more attention paid to interfaces that are novel and technologically sophisticated instead of interfaces that actually help users.

User interfaces should be appropriate for the job at hand. No one user interface is appropriate for all applications, although frequently it is easier to adapt to an interface that exists than to develop an entirely new interface.

The ubiquitous character-oriented user interface, which depends upon the keyboard and alphanumeric display, was fashioned after typewriters. It is most appropriate for word processing, although it has been adapted to almost everything regardless of how inappropriate and hard-to-use the result was.

The first graphical user interface computer is usually regarded as the Xerox Star. In the mid 1980's Apple Computer introduced its Lisa and Macintosh series of computers. One of the main developments of this was that a social atmosphere was created whereby line-oriented user interfaces were shunned, forcing developers to learn how to integrate the graphical user interface into their applications. The results were a much more user-oriented environment. Such technology is now being ported to other graphical user interfaces, such as the X Window System.

Software developers and users need to remember that a graphical user interface alone does not automatically make software easier to use, although it does allow greater opportunities. You cannot make software easy to use simply by linking in the "ease of use" library.

User interface is a subtle art. Making software easy to use has as much to do with what is omitted as with what is included.

Even power users need good user interfaces. Power users are intelligent people whose time is valuable. Although they have a greater capacity for details and a greater tolerance for bad user interfaces, we will make the greatest gains by expediting their work. It may be true that the user interfaces that are best for their purposes are different from those that are appropriate for beginners, but that does not mean that they should be content with spartan user interfaces.

As most users are not well versed in user interface techniques, the bulk of the user interfaces will come from commercial products, which will be programmed in their own languages by users for specific tasks.

User Interfaces for Scientific Computing

Numerical, Symbolic and Graphical capabilities are or soon will be available to all simultaneously. For the solution of large problems, probably all capabilities will be used. The increased complexity of these tools calls for improved user interfaces.

Large computing on supercomputers have been characterized by massively powerful processing on huge data sets, whereas personal computers have been characterized by improved user interfaces. These technologies are cross-fertilizing as workstation power increases and as graphical user interfaces and user interface techniques migrate to larger systems.

One of the reasons why smaller computers have been easier to use has been sheer simplicity. If the machine has fewer moving parts, it's going to be easier to understand the parts that must be manipulated and therefore the user will have more control and therefore more power, even though the machine is inherently less powerful.

On larger computers it is typical for designers to incorporate more commands, options, and facilities. The resulting mind-boggling array of choices, and the subsequent confusion caused by their use, is one of the biggest reasons why larger computers are falling under disfavor. Designers think that they should be moving in the direction of freedom of choice, but instead they should be moving in the direction of freedom from choice.

One of the tricky arts of user interface design is to hide complexity behind software that automatically takes care of the details. This can be a big problem with mathematical software. Everywhere we turn, we find exceptional situations that cause our algorithms to break, requiring human oversight and intervention.

For instance, simple integration algorithms using fixed step sizes worked well, except for functions with singularities or other violent irregularities. Users of such software had to be mindful of such problems, paying dearly in attention-span resources, and in some cases adjusting options until satisfactory results were achieved. Newer algorithms that use adaptive stepsizes are more robust in those respects. Developers of such algorithms have tended to think that they were increasing the reliability of the algorithm; another way to look at it is that they were improving its user interface by taking care of details for the user automatically, searching for irregularities and doing the right thing for each instance, so that the user has more attention span to devote to his work.

One element that is needed to increase the state of the art is for air- tight algorithms that "always" work, or that can notify a user of their failure in the case that they don't work. What is NOT needed is algorithms that fail by handing back an incorrect answer with no other notification, as these will end up costing the user much more time than the presumably speedy algorithm saved in the first place.

Developers of scientific software will increasingly have to view the user as a stubborn and naive all-terrain vehicle driver who is willing to drive in any direction, through the mud and up sheer edifices persistently until either he arrives at his goal, or his vehicle overturns, whereupon he will right his vehicle and look for another path to his goal, perhaps without really understanding the internals of the vehicle or the reasons for the failure. The developers who can build such vehicles will find their creations more popular than those who don't.

User Interfaces for Programming

There is a common belief that interpreted programming languages are easier to use than compiled languages. One reason for this is that interpreted languages tend to have more runtime checks built into their interpreters.

Another reason is that the edit-compile-run loop is faster. Programming is a trial-and-error process that obviously accelerates as this loop is tightened up. In addition, studies have shown that fast response to input is crucial for maintaining the attention span of the user. In other words, the longer your compiler takes, the better the chance that you will become distracted and loose track of the details of your programming task.

These advantages need not be available only to those with interpreted programming environments; ultra-fast compilers, incremental compilation, and runtime consistency checks can achieve the same results. As the performance of our machines skyrockets relative to the cost of our attention span, we must recalculate the trade off between getting answers slowly and getting answers incorrectly.

Programmers and other scientific users need higher level tools. It is no longer acceptable for us to write a do-loop to add two vectors, because of the minuscule chance for error in such code, and the added attention span we must devote to the extra code every time we examine it. Using higher-level commands to add vectors and do other high-level operations not only result in increased programmer productivity but also allow for vector processors and other optimization schemes to be more readily utilized. In addition, if the statements in your programming language are higher-level, the penalties for interpreted programming diminish as the ratio between interpreter overhead and the intended processing falls.

Good User Interface Technology

This section goes over some user interface techniques that will become more and more common as the technology advances.

Realistic Equation Display

Since the dawn of computing, scientists have been entering equations into computers. Almost always this has been in a purely textual format. Recently more realistic, two dimensional equation display has been available on graphical workstations.

Expressions are more difficult to display than simple text. Text, especially in a single fixed-pitch font, grows lengthwise with each character and only rarely grows in height, and then, only when automatic word-wrap is enabled. Formulas, on the other hand, can grow and shrink vertically and horizontally with each keystroke as subscripts, superscripts and fractions change.

The expression is usually represented internally as a tree. Each node contains coordinates specifying a rectangle that encloses the expression that the node encloses, in addition to other parameters such as math axis, baseline, font size, etc. The drawing of an expression usually is a two-step process.

The first pass traverses the tree and calculates the size of each node in the tree. Each node needs the size of each sub-expression tree; the algorithm is post-order recursive. The sizing algorithm must line up consecutive symbols so that their math axes line up; in other words, centers of plus and minus signs must be at the same height as fraction lines.

The second pass actually draws each node. The algorithm can be either pre-order or post-order.

Direct Manipulation

The advent of graphical workstations has put a mouse or other graphical input device on everybody's desk. Unfortunately, platforms that have their roots in alphanumerics have had a slow transition to making good use of the mouse.

The typical workstation has a quarter million to a million pixels on its screen. Each is a valid location for the mouse. When the user clicks and drags the mouse, he supplies 32 to 40 bits of information to the computer, whereas each keystroke is only about six bits of information. Clearly, the mouse can be a much faster input device than the keyboard.

More important, though, is the kind of information it supplies. The mouse almost always is used to point to something that has been drawn on the screen. As such the available functions can be changed as fast as the screen can be redrawn. In addition, pertinent objects in the software can be implicated by the user without resorting to assigning numbers or names to them and forcing the user to type them in.

An example of this is an equation editor. The user can click and drag the mouse and select a sub-expression in an equation far more easily than ever possible with a keystroke-based mechanism, which usually resorted to describing the path down a tree that must be taken to find the given sub-expression.

Another example of this is performing symbolic algebra operations by clicking and dragging in Theorist. Selected sub-expressions can be dragged around on the screen to rearrange equations, and whole equations can be dragged to other equations for substitution. Using an alphanumeric approach, equations would be numbered with arbitrary labels and the user would type such labels into commands to achieve similar results.

Much richer are the opportunities for direct manipulation of graphs. One of the most common is rotation of a 3D graph by clicking and "rolling". Less obvious are some of the functions founa, for instance, in Spyglass View, where you can modify a color look up table by clicking and dragging along a spectrum displayed along the edge of a window.

Click-and-drag code is usually implemented in a loop or a set of routines that respond to events that the application is sent from the windowing system. While the mouse button is held down, the application program must provide some sort of feedback, called a "ghost", as to what would happen if the mouse were released at the given position. Sometimes, if the workstation is powerful enough, the action is completely taken care of as the user clicks and drags, but this is not always practical.

Given the notification that the user is clicking in such a location, on a Macintosh, the code looks like this:

```
Get current mouse position
Draw Ghost according to mouse position
while (user holds mouse down)
        Get current mouse position
        if mouse position implies a different ghost,
                Undraw old Ghost
                Draw new Ghost
        end loop
Undraw old Ghost
Perform action, if not canceled
```

It is always good to provide some way for the user to harmlessly cancel a click-and-drag in the middle, such as dragging to an illegal location.

On most other windowing platforms, the application writer does not have the luxury of writing his own loop but is allowed to provide the prelude, the body of the loop, and the post-processing as three separate procedures that are called by the windowing system. The results are the same although the code is usually less clear.

On workstations that have a simple one bit deep screen, the ghost is frequently drawn in exclusive-Or mode, so that it can be undrawn with the same code. An alternative is to save the pixels underneath the ghost for subsequent replacement during undrawing.

Automatic Recalculation

One of the most common programs in use today for scientific analysis is the spreadsheet. You type some numbers into certain cells, and you type in some formulas into other cells. The cells with the formulas will display the results of the calculations, which use the numbers as inputs. Formulas can be chained endlessly for arbitrarily complicated calculations. When you change any numbers or formulas, the derived cells are recalculated automatically. If the calculations are simple, the screen is updated immediately, providing instant feedback to the user.

Many spreadsheets can draw graphics. The graphics are the result of calculations from formulas, and the graphs are updated automatically after any change is made to any numbers, formulas, or any controls that are found on the graph itself that the program supplies.

A modification of this can be found in programs such as MathCAD and Theorist. In these programs, the formulas are more central and therefore are not hidden behind the numbers that are the result of their being calculated. Also, the formulas are displayed in a realistic, 2D format as formulas are displayed in books. Any change to formulas or numbers is reflected in the graphs or number displays as quickly as the computer can recalculate the answers.

Obviously, this can be extended to other forms of "calculations" and "formulas" and "displays". For instance, programs exist that display a schematic diagram of an analog circuit. Signals can be fed into the circuit by the user and the output observed on software "oscilloscopes", or the output can be heard when the computer feeds it out of a DAC and a speaker. The output changes instantly to reflect any changes in setup or input.

Similarly, video image processing or symbolic operations can be performed in this non-procedural way, as long as some "formula" or recipe for deriving the result is readily available and comprehensible to the user.

The two problems with this technique are:

- Sometimes recalculation is painfully slow.
- Properly retriggering recalculation can be tricky.

The most common way to correctly trigger recalculation is to maintain dependency information, frequently in a tree form. For instance, graph A depends upon formula 12 which depends upon formula 47. If formula 47 is changed by the user, that must trigger the invalidation of formula 12, which in turn triggers the invalidation of graph A. Generally you have a set of pointers that point in both directions; one direction is used for calculation and the other for invalidation. These pointers must be kept up to date as the formulas and dependencies change.

The most common way to help recalculation speed is to cache intermediate results. For instance, in Theorist, a three-dimensional graph has two levels of cache. The data cache contains coordinates of points and a three dimensional description of what is to be drawn. An image cache contains a pure image of the graph. If the window is obscured by another window and later exposed, only the image cache need be accessed. If the graph is rotated to a different angle by user command, the image cache is invalidated, forcing a redraw from the data cache, which is still valid. If the equations change, though, both z are invalidated triggering a complete recalculation.

The more caches there are, the more time can be saved in recalculation. On the other hand, this also makes the software more complex and uses more memory. The complexity is especially acute; frequently it is simpler to conservatively invalidate more than is necessary than to perfect a multi-tiered cache system. Failure to correctly invalidate is a particularly odious problem.

Conclusion

The scientific computing systems of the future will require more mature user interfaces in order to fully utilize the advanced tools that will be available, in addition to having more robust tools available. Almost always these will involve commercial software packages.

SENAC: Lisp as a Platform for Constructing a Problem Solving Environment

by

Kevin A. Broughan
Department of Mathematics and Statistics
University of Waikato
Hamilton
New Zealand
kab@waikato.ac.nz

Abstract

This lecture will provide an introduction to the SENAC problem solving environment by reviewing the advantages and disadvantages of the language Lisp for the development of high-level features based on the experience of the Mathematical Software Project at the University of Waikato.

In particular the features of Common Lisp which are advantageous for environment development, the use of foreign function interfaces for linking to subroutine libraries, the use of interprocess communication for driving graphics systems, the role of common storage and callbacks, ideas for improved portability and the introduction of parallelism will be among the more classical topics developed in the context of a application.

Complex application of graphical, symbolic and numerical methods in packages for solving mathematical modeling problems

I.A. Nicolayev, L.A. Krukier, S.A. Zharinov

Rostov State University, USSR

The widespreading of computers and their application field expansion makes it actual the development of mathematical software for non-experts in mathematical modeling and programming. One of the main tasks for this aim is a simplicity and visuality of information input and output with help of the graphical instruments and methods. The symbolic methods usage for mathematical modeling problems makes it possible to simplify program structure and extend its flexibility and resources. The effective numerical methods permit computers to solve the application problems quickly and accurately. So graphical, symbolic and numerical methods complex application for mathematical modeling is essential for software quality improvement.

One of the most popular approaches for computer mathematical modeling is development of intellectual application program packages intended for the solutions of the problems from certain field of human activity. The modern packages are based on the friendly user's interface adapted to different user's knowledge levels and include some elements of expert systems and knowledge bases.

In the framework of this approach PACEPACK (Parallel Algorithms for Computing Elliptic PACKage) package is developed in Computing Center of Rostov University. The package is a software system for solving two-dimensional elliptic boundary value problems with arbitrary boundary conditions and is realized on complex PC - multiprocessor. The package structure in general is similar to ELLPACK [1] but has some essential features. Instead of high-level problem statement language in ELLPACK there is a menu with equation templates set, domain and boundary input is accomplished by scanner or mouse device, finite differences approximations of differential operator are performed by computer algebra system, solution module choice and output representation form are controlled by expert system. These peculiarities improve package abilities and make it easy of access for a wide variety of users.

Boundary value problems solving includes two main step. The first step is the partial differential equation and domain preparation, and the second one is numerical solving of resulting linear algebra equations system. On the first step there is a problem specification dialogue so it seems to be reasonable to perform it on a personal computer. For most real world problems resulting linear algebra equations system is very complicated and fast processing is required to solve it. So the second step is performed on multiprocessor.

In order to describe PACEPACK architecture let us consider the process of the solving elliptic boundary problem (fig. 1) using one of the methods of discretization.

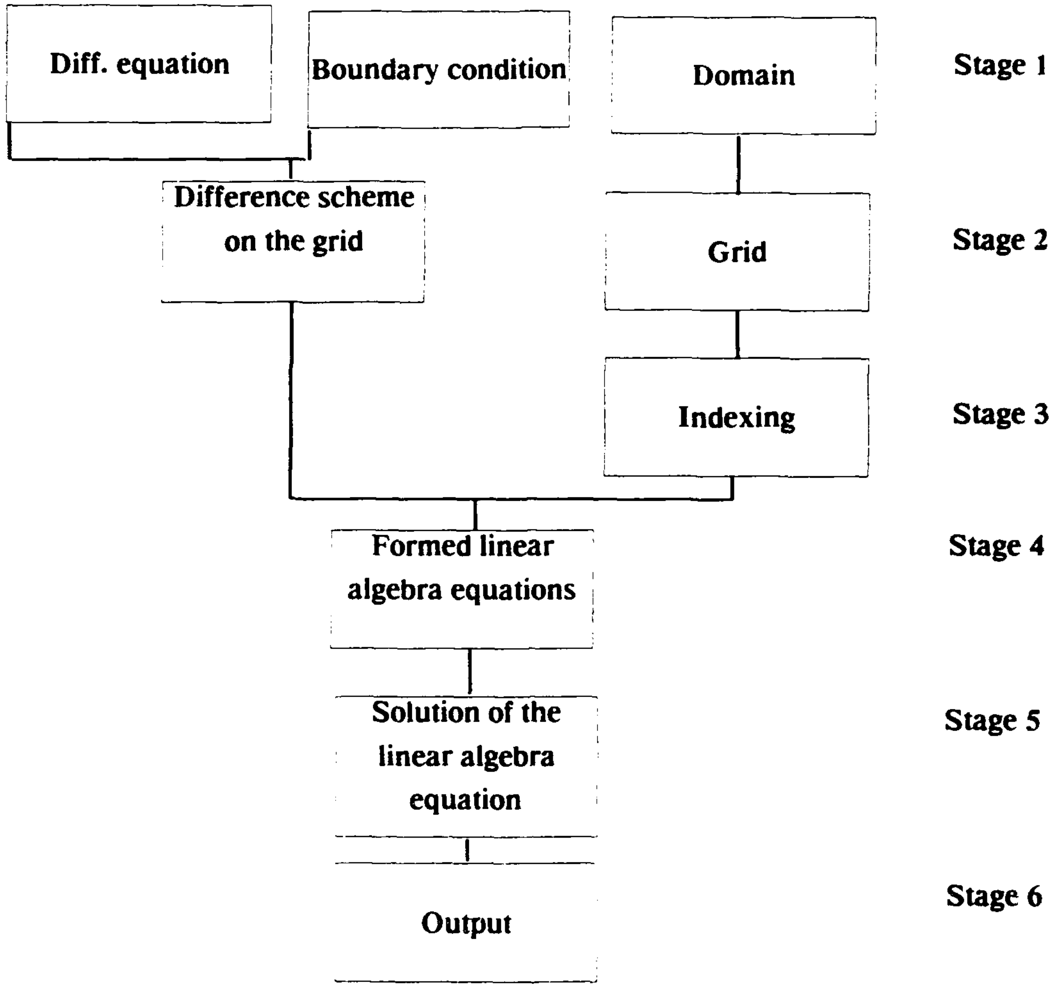

Diff. equation
Boundary condition
Domain
Stage 1
Difference scheme on the grid
Grid
Stage 2
Indexing
Stage 3
Formed linear algebra equations
Stage 4
Solution of the linear algebra equation
Stage 5
Output
Stage 6

Fig. 1

The elliptic equation in two dimensions domain Ω (the first stage) has the form

$$(aU_x)_x + (bU_x)_y + (cU_y)_x + (dU_y)_y + (eU)_x + (fU)_y + nU = g \qquad (1)$$

with the following boundary conditions on $\partial\Omega$

$$p_1U_x + p_2U_y + qU = r \qquad (2)$$

where coefficients in (1) and (2) are the function of x, y. Unknown U and functions g, r may be vectors; a, b, c, d, e, f, n, p_1, p_2, q may be matrixes, but the elliptic conditions must be contended in any cases. The second stage of the task is a construction of the grid in domain Ω and difference approximation of the equation (1) and boundary conditions (2). Then difference scheme on the grid is produced. Indexing of nodes in the grid (the third stage) gives the structure of the linear system of equations(the forth stage), and solution the system is the fifth stage.

The relatively indepedence of the stages for fixing input-output gives the opportunity to utilize the modules, which were developed and tested before. This approach makes it possible to construct the packages for solving an elliptic problem quickly enough. The usage of a personal computer with the multiprocessor board gives an opportunity to make an effective utilization of rich software which was produced for PC.

Each of these stages for solving a problem can be realized as a block of the package. Now we consider these blocks. Let us examining ones in details.

Problem description block. The block includes tools to input information of the problem into computer, that is software tools have to grasp, recognize and analyze input information. The block is the most informative one because of it transfers a lot of the problem into computer. Input information is divided with the following manner: control information to run the problem and information of data problem. We must point out that information of data problem initiates the part of control information. Transferring of the problem into computer is executed in dialogue manner. User has the opportunity to return into the block after any block of the package finished its work to correct the program without its recompiling.

The most complicated part of the block is the program which forms a computer representation of the complex geometry domain with cuts and holes. The program produces a bound configuration with analytical methods and point-wise method.

Grid configuration block. This block transforms continuous space into set of regular nodes, which connected with arrays of coefficients, unknowns, right parts. Initial and boundary conditions and partial derivatives should be approximated. The grid construction in regular domain isn't a complicated problem. But this procedure in irregular domain is a complex one. It has to be realized on multicomputer architecture if adaptive grid will be constructed.

In order to match grid configuration on monitor screen with its image in computer memory another problem arises. The program module is developed to fulfil correction of a grid during construction period and after the grid is constructed. The module uses information of domain geometry from the previous block and can interact with user by means of dialogue. The following analyse with the help of the expert system makes an opportunity to choose the constructing grid algorithm. Output information of the block consists of grid point coordinates (inner and outer points of boundary) and array of boundary points.

These two blocks were fulfilled as a separate program file using PASCAL 5.0. The part of the second block for grid construction was developed by using the package GRAPFHIX TOOL BOX 4.0.

Block of approximation of equation. The block explores algorithm of finite differences approximation of linear operator on the base of undefined coefficients method. Equation (1) is replaced by

$$\sum_{i,j \in T} C_{i,j}\, U(x + i\, h,\;\; y + j\, h) \qquad (3)$$

where U(x + i h, y + j h) is function approximation on the grid, $C_{i,j}$ - coefficients to be defined, T - some template on the grid, h - step of the grid.

Coefficients $C_{i,j}$ are defined by power expansion of expression (3) and comparing them with exprassions (1), (2). To solve the resulting system of linear equations original method is used.

The approach is implemented on the base of LISP with help of some elements of the computer algebra system mu-MATH. User has the possibility to control template choice and approximation precision through dialog.

Block of linear algebra system composition. The block determines the order in array of grid points of the domain and as a result of the procedure is a spare matrix of linear system. The block isn't a significant part of the package if user solves the problem on single processor computer. Algorithms included into the block are used to solve the problem with finite-element method which reduces the problem into linear system with any sparse matrix [2]. A most parts of algorithms in ELLPACK package are developed to make band of linear system matrix as tight as possible.

The block becomes significant part of the package when multiprocessor computer is being used for solving the problem. The main object of the block is a matching matrix structure to the computer architecture [3,4].

Block for solving linear system equations. Input information of the block is matrix characteristics (positive or negative form, self-adjoint structure, coefficients and etc.) and output of the block is a vector of unknowns. The user can point out an advisable method of solving linear algebra system: iterating or direct one. Algorithms which are realized in the block can run on a single or multi-processor. Then the expert system uses information of the multi-processor architecture to form matrix structure of the system which is matched to the architecture. We mention to the module analysis of the algorithms [5] where it was said that the efficiency of the parallel algorithms depended on the pseudo-residual method of calculations.

Output block. The solution of linear equation system is passed from multi-processor to PC, where information about grid and nodes indexing were using is stored. The complex analyze of the data gives opportunity to construct graphics of the behaviour of the solution in the domain on screen or printer.

So complex application of graphical, symbolic and numerical methods in packages makes it possible to solve in direct manner many complicated problems of designing and exploiting software for science and applications.

References:

1. Rice J.R., Boisvert R.F. Solving Elliptic Problems Using ELLPACK - Berlin, Springer Verl, 1985.

2. Reid J.K. Algebraic aspects of finite-element solution Comp. Phys Rept, 1987, 6, N 1-6, p. 385-414.

3. Adams L.M. Iterating Algorithms for Large Sparse Linear Systems on Parallel Computers. NASA Contractor Report, N 166027, 1982.

4. Liu J.W. Reordering-Sparse matrixes for parallel elimination Paral. Comp., 1989, V II, N 1, p. 73-91.

5. Krukier L.A., Simonovich I.V. Computing Systems and Algorithms, Rostov University, USSR, 1983 (on Russian).